MCUXpresso SDK Documentation

Release 26.09.00

NXP
Sep 29, 2026

Table of contents

1 MIMXRT700-EVK **3**
- 1.1 Overview . . . 3
- 1.2 SDK Introduction . . . 3
  - 1.2.1 MCUXpresso Software Development Kit (SDK) . . . 3
- 1.3 Getting Started with MCUXpresso SDK Package . . . 7
  - 1.3.1 Getting Started with Package . . . 7
- 1.4 Getting Started with MCUXpresso SDK GitHub . . . 30
  - 1.4.1 Getting Started with MCUXpresso SDK Repository . . . 30
- 1.5 Getting Started with MCUXpresso SDK Xplorer . . . 57
  - 1.5.1 Getting Started with MCUXpresso SDK Xplorer . . . 57
- 1.6 Release Notes . . . 73
  - 1.6.1 MCUXpresso SDK Release Notes . . . 73
- 1.7 ChangeLog . . . 78
  - 1.7.1 MCUXpresso SDK Changelog . . . 78
- 1.8 Driver API Reference Manual . . . 192
- 1.9 Middleware Documentation . . . 192
  - 1.9.1 VG-Lite GPU Library . . . 192
  - 1.9.2 Multicore . . . 192
  - 1.9.3 MCU Boot . . . 192
  - 1.9.4 eIQ . . . 192
  - 1.9.5 FreeMASTER . . . 192
  - 1.9.6 NXP Wi-Fi . . . 192
  - 1.9.7 FreeRTOS . . . 192
  - 1.9.8 lwIP . . . 192
  - 1.9.9 File systemFatfs . . . 193
  - 1.9.10 DSP Audio Streamer . . . 193
  - 1.9.11 Trusted-Frimware-M . . . 193

2 MIMXRT798S **195**
- 2.1 ACMP: Analog Comparator Driver . . . 195
- 2.2 CACHE: CACHE Memory Controller . . . 202
- 2.3 CDOG . . . 206
- 2.4 Clock Driver . . . 210
- 2.5 CRC: Cyclic Redundancy Check Driver . . . 267
- 2.6 CTIMER: Standard counter/timers . . . 271
- 2.7 Dsp . . . 281
- 2.8 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver . . . 282
- 2.9 eDMA core Driver . . . 316
- 2.10 eDMA soc Driver . . . 325
- 2.11 Ezhv . . . 326
- 2.12 FGPIO Driver . . . 330
- 2.13 FlexIO: FlexIO Driver . . . 330
- 2.14 FlexIO Camera Driver . . . 330
- 2.15 FlexIO Driver . . . 334
- 2.16 FlexIO eDMA I2S Driver . . . 354
- 2.17 FlexIO eDMA MCU Interface LCD Driver . . . 358

2.18 FlexIO eDMA SPI Driver . . . 360
2.19 FlexIO eDMA UART Driver . . . 364
2.20 FlexIO I2C Master Driver . . . 367
2.21 FlexIO I2S Driver . . . 376
2.22 FlexIO MCU Interface LCD Driver . . . 388
2.23 FlexIO SPI Driver . . . 401
2.24 FlexIO UART Driver . . . 415
2.25 FREQME: Frequency Measurement . . . 427
2.26 GDET . . . 427
2.27 Glitch Detect . . . 429
2.28 GLIKEY . . . 429
2.29 GLIKEY . . . 429
2.30 GPIO: General-Purpose Input/Output Driver . . . 434
2.31 GPIO Driver . . . 437
2.32 I3C: I3C Driver . . . 443
2.33 I3C Common Driver . . . 446
2.34 I3C Master Driver . . . 449
2.35 I3C Master DMA Driver . . . 480
2.36 I3C Slave Driver . . . 483
2.37 I3C Slave DMA Driver . . . 497
2.38 INPUTMUX: Input Multiplexing Driver . . . 500
2.39 Iopctl_driver . . . 513
2.40 IRTC: IRTC Driver . . . 516
2.41 ITRC . . . 525
2.42 Intrusion and Tamper Response Controller . . . 529
2.43 JPEGDEC: JPEG decoder Driver . . . 529
2.44 Common Driver . . . 538
2.45 LCDIF: LCD interface . . . 557
2.46 LPADC: 12-bit SAR Analog-to-Digital Converter Driver . . . 583
2.47 Lpc_freqme . . . 604
2.48 LPFLEXCOMM: LPFLEXCOMM Driver . . . 610
2.49 LPFLEXCOMM Driver . . . 610
2.50 LPI2C: Low Power Inter-Integrated Circuit Driver . . . 612
2.51 LPI2C Master Driver . . . 613
2.52 LPI2C Master DMA Driver . . . 628
2.53 LPI2C Slave Driver . . . 630
2.54 LPSPI: Low Power Serial Peripheral Interface . . . 642
2.55 LPSPI Peripheral driver . . . 642
2.56 LPSPI eDMA Driver . . . 666
2.57 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver . . . 672
2.58 LPUART Driver . . . 672
2.59 LPUART eDMA Driver . . . 695
2.60 MIPI DSI Driver . . . 699
2.61 MIPI_DSI: MIPI DSI Host Controller . . . 722
2.62 MMU: Memory Map Unit . . . 722
2.63 MRT: Multi-Rate Timer . . . 734
2.64 MU: Messaging Unit . . . 739
2.65 This type defines status return values used by NBOOT functions that are not easily . . . 752
2.66 OSTIMER: OS Event Timer Driver . . . 753
2.67 PDM: Microphone Interface . . . 758
2.68 PDM Driver . . . 758
2.69 PDM EDMA Driver . . . 771
2.70 PINT: Pin Interrupt and Pattern Match Driver . . . 776
2.71 PNGDEC: PNG decoder Driver . . . 785
2.72 Power Driver . . . 791
2.73 PUF: Physical Unclonable Function . . . 820
2.74 Puf_v3_driver . . . 820
2.75 Pvts . . . 826

2.76 Reset Driver . . . 829
2.77 Runbootloader . . . 841
2.78 SAI: Serial Audio Interface . . . 842
2.79 SAI Driver . . . 842
2.80 SAI EDMA Driver . . . 872
2.81 Sbloader . . . 879
2.82 SCTimer: SCTimer/PWM (SCT) . . . 884
2.83 SDADC: SDADC Module . . . 908
2.84 SEMA42: Hardware Semaphores Driver . . . 924
2.85 Soc_mipi_dsi . . . 928
2.86 SYSPM: System Performance Monitor . . . 928
2.87 TRNG: True Random Number Generator . . . 931
2.88 USDHC: Ultra Secured Digital Host Controller Driver . . . 937
2.89 UTICK: MictoTick Timer Driver . . . 972
2.90 WWDT: Windowed Watchdog Timer Driver . . . 973
2.91 CACHE: CACHE Memory Controller . . . 977
2.92 XSPI: Serial Peripheral Interface Driver . . . 980
2.93 XSPI eDMA Driver . . . 1048

3 Middleware **1053**
3.1 Boot . . . 1053
3.1.1 MCUboot in MCUXpresso SDK . . . 1053
3.2 Connectivity . . . 1054
3.2.1 lwIP . . . 1054
3.3 File System . . . 1068
3.3.1 FatFs . . . 1068
3.4 Trusted Firmware-M . . . 1070
3.4.1 Trusted-Firmware-M . . . 1070
3.5 Motor Control . . . 1523
3.5.1 FreeMASTER . . . 1523
3.6 Multicore SDK . . . 1561
3.6.1 Quick links . . . 1561
3.6.2 Components . . . 1562
3.7 Multimedia . . . 1651
3.7.1 VGLite Graphics Driver . . . 1651
3.7.2 Xtensa Audio Framework (XAF) . . . 1743
3.8 Wireless . . . 1757
3.8.1 NXP Wireless Framework and Stacks . . . 1757

4 RTOS **1819**
4.1 FreeRTOS . . . 1819
4.1.1 FreeRTOS kernel . . . 1819
4.1.2 FreeRTOS drivers . . . 1823
4.1.3 backoffalgorithm . . . 1823
4.1.4 corehttp . . . 1823
4.1.5 corejson . . . 1823
4.1.6 coremqtt . . . 1823
4.1.7 corepkcs11 . . . 1824
4.1.8 freertos-plus-tcp . . . 1824

5 Downloads **1825**
5.1 boards . . . 1825
5.2 develop . . . 1825

Bibliography **1827**

Index **1829**

This documentation contains information specific to the mimxrt700evk board.

Chapter 1

MIMXRT700-EVK

1.1 Overview

The i.MX RT700 EVK (MIMXRT700-EVK) features NXPs advanced implementation of the Arm Cortex-M33 core, combined with the highly optimized Cadence Tensilica HiFi4 and HiFi1 DSP processor cores. MIMXRT700-EVK supports development for the MIMXRT798, MIMXRT758 and MIMXRT735 products and its features make it ideal for portable HMI applications. The i.MX RT700 EVK can help jump start your next design with the included schematics and layout files.

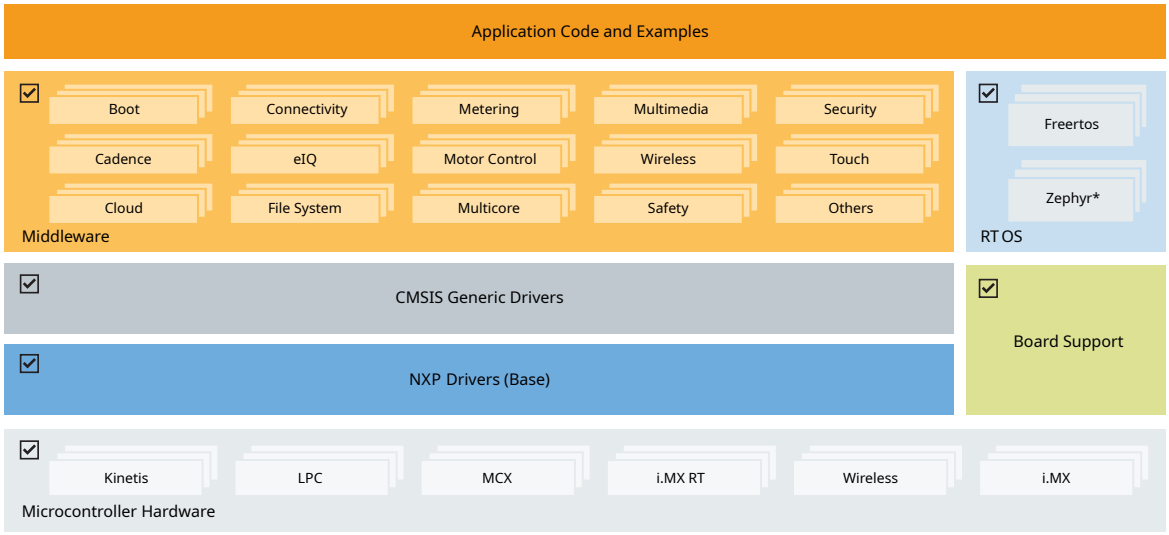
MCU device and part on board is shown below:

- Device: MIMXRT798S
- PartNumber: MIMXRT798SGFOB

1.2 SDK Introduction

1.2.1 MCUXpresso Software Development Kit (SDK)

The MCUXpresso SDK is a comprehensive software enablement package designed to simplify and accelerate application development with Arm® Cortex®-M based devices from NXP, including its general purpose, crossover and wireless-enabled MCUs. It includes production-grade software with integrated enabling software technologies (stacks and middleware), reference examples, and more.



*Zephyr RT OS leverages base SDK components via NXP Zepher HAL

Features

Comprehensive Driver Library

- Flexible peripheral drivers for rapid development
- Optimized for performance and code size

RTOS Integration

- FreeRTOS support with optimized configurations
- Real-time application frameworks
- Task management and scheduling examples

Middleware Components

- Connectivity stacks (USB, Ethernet, wireless, communication protocols)
- AI inference engines and neural network libraries
- Audio processing and voice recognition libraries
- Security frameworks and cryptographic libraries
- Graphics and user interface components
- Motor control and real-time control libraries
- File systems and storage solutions
- Partner middleware can be added using CMake and west manifests

Extensive Examples

- Basic peripheral use cases to full technology demonstrations
- Board-specific reference implementations

- Out-of-box Demo Applications
- Middleware component examples and integrations

Distribution Formats

MCUXpresso SDK is available in two main formats:

GitHub Repository SDK Git-based distribution hosted on GitHub for modular development, continuous updates, and collaborative workflows. The SDK uses multiple repositories managed by the Zephyr west tool to provide flexible, component-based development. See Getting Started for setup instructions.

SDK Package Downloadable packages from SDK Builder for offline development. Starting with release 25.12.00, two package versions are available:

- **Classic SDK Package**: Traditional board-specific packages with pre-configured IDE projects for MCUXpresso IDE, IAR, Keil, and other toolchains.
- **Repository-Layout SDK Package**: Board-specific packages that maintain the same structure and build system as the GitHub Repository SDK, providing offline access to the repository-based development experience. Available when selecting the ARMGCC toolchain.

Note: From version 25.12.00 onward, selecting ARMGCC downloads the Repository-Layout version; all other toolchains remain in the Classic version.

Delivery and Release

NXP integrates Coverity® Static Analysis and Black Duck Software Composition Analysis into the SDK development process to deliver secure, high-quality software.

- **Annual releases (June)** for popular products, with Long Term Support (LTS) on GitHub — back-ported security (SVE/PSIRT) and severe bug fixes for two years.
- **Quarterly releases** for NPI and recent products.
- SBOMs provided for all GitHub SDK repositories.

Release	Release Name	Maintenance
LTS	26.06.00-LTS	2 years until June 2028
Quarterly	26.09.00	None, rebase to mainline
LTS	27.06.00-LTS	2 years until June 2029

Supported platforms are listed at mcuxpresso.nxp.com/sdk-versions.

Toolchain Support

All driver examples and most application examples are provided for the following toolchains:

Toolchain	GitHub	Archive/Zip	Generated (west)	Getting Started
MCUXpresso IDE	✅	✅ eclipse	✅	Link
MCUXpresso for VS Code	✅	✅ CMake	N.A.	*Link*
IAR Embedded Workbench	✅	✅ EWARM	✅ EWARM	Link
Keil™ MDK-Arm	✅	✅ uVision	✅ uVision	Link
CLI GNU toolchain	✅	✅ CMake	N.A.	*Link*

Archive/Zip packages can be downloaded from SDK Builder or imported directly in VS Code and MCUXpresso IDE using the Import Wizard.

IAR and Keil projects can be generated from the GitHub SDK CMake examples using a west build command.

Repository Architecture

The SDK uses a multi-repository structure on GitHub managed by the Zephyr West Tool, enabling:

- **Modular Organization**: Drivers, RTOS, middleware, and examples in separate repositories
- **Customizable Manifests**: Targeted setups for specific requirements
- **Selective Downloads**: Include only needed components
- **Independent Updates**: Component-level version control
- **Scalable Integration**: Easy addition of new repository components

Support and Resources

Community Support

- Open GitHub issues and discussions for questions and contributions
- MCUXpresso SDK Community Forum
- MCUXpresso Software and Tools Forum

Documentation

- Comprehensive guides and API documentation online

Technical Support

- NXP Support for commercial development assistance

SDK Product Page

- MCUXpresso SDK on NXP.com

Trainings

- MCUXpresso Training Hub
- MCUXpresso SDK Curriculum

1.3 Getting Started with MCUXpresso SDK Package

1.3.1 Getting Started with Package

MCUXpresso SDK board support package folders

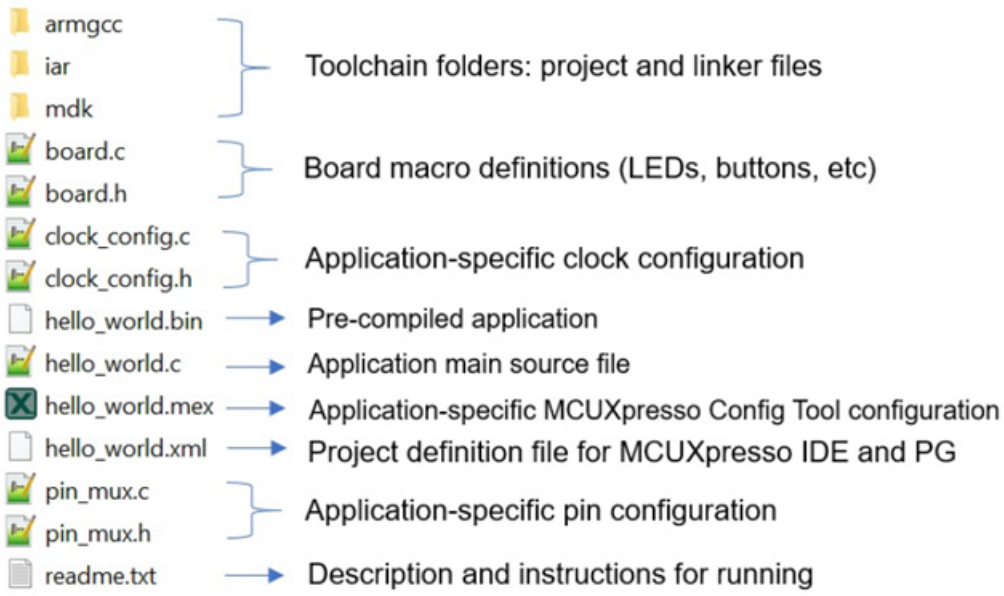
MCUXpresso SDK board support package provides example applications for NXP development and evaluation boards for Arm Cortex-M cores including Freedom, Tower System, and LPCXpresso boards. Board support packages are found inside the top-level boards folder and each supported board has its own folder (an MCUXpresso SDK package can support multiple boards). Within each <*board_name*> folder, there are various subfolders to classify the type of examples it contains. These include (but are not limited to):

- cmsis_driver_examples: Simple applications intended to show how to use CMSIS drivers.
- demo_apps: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- driver_examples: Simple applications that show how to use the MCUXpresso SDK peripheral drivers for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI conversion using DMA).
- emwin_examples: Applications that use the emWin GUI widgets.
- rtos_examples: Basic FreeRTOS OS examples that show the use of various RTOS objects (semaphores, queues, and so on) and interfaces with the MCUXpresso SDK RTOS drivers.
- usb_examples: Applications that use the USB host/device/OTG stack.

Example application structure This section describes how the various types of example applications interact with the other components in the MCUXpresso SDK. To get a comprehensive understanding of all MCUXpresso SDK components and folder structure, see *MCUXpresso SDK API Reference Manual*.

Each <*board_name*> folder in the boards directory contains a comprehensive set of examples that are relevant to that specific piece of hardware. Although we use the hello_world example (part of the *demo_apps* folder), the same general rules apply to any type of example in the <*board_name*> folder.

In the *hello_world* application folder, you see the following contents:



All files in the application folder are specific to that example, so it is easy to copy and paste an existing example to start developing a custom application based on a project provided in the MCUXpresso SDK.

Parent topic:*MCUXpresso SDK board support package folders*

Locating example application source files When opening an example application in any of the supported IDEs, various source files are referenced. The MCUXpresso SDK devices folder is the central component to all example applications. It means that the examples reference the same source files and, if one of these files is modified, it could potentially impact the behavior of other examples.

The main areas of the MCUXpresso SDK tree used in all example applications are:

- *devices/<device_name>*: The device's CMSIS header file, MCUXpresso SDK feature file, and a few other files
- *devices/<device_name>/cmsis_drivers*: All the CMSIS drivers for your specific MCU
- *devices/<device_name>/drivers*: All of the peripheral drivers for your specific MCU
- *devices/<device_name>/<tool_name>*: Toolchain-specific startup code, including vector table definitions
- *devices/<device_name>/utilities*: Items such as the debug console that are used by many of the example applications
- *devices/<devices_name>/project*: Project template used in CMSIS PACK new project creation

For examples containing an RTOS, there are references to the appropriate source code. RTOSes are in the *rtos* folder. The core files of each of these are shared, so modifying one could have potential impacts on other projects that depend on that file.

Parent topic:*MCUXpresso SDK board support package folders*

Run a demo using MCUXpresso IDE

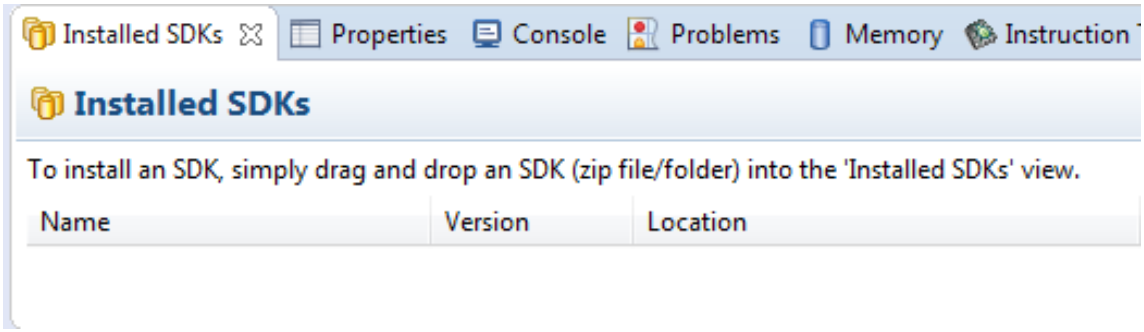
This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The hello_world demo application targeted for the target hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

Select the workspace location Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse, which uses the workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace. The location of the workspace can be anywhere, but it is recommended that the workspace locate outside the MCUXpresso SDK tree.

Parent topic:*Run a demo using MCUXpresso IDE*

Build an example application To build an example application, perform the following steps.

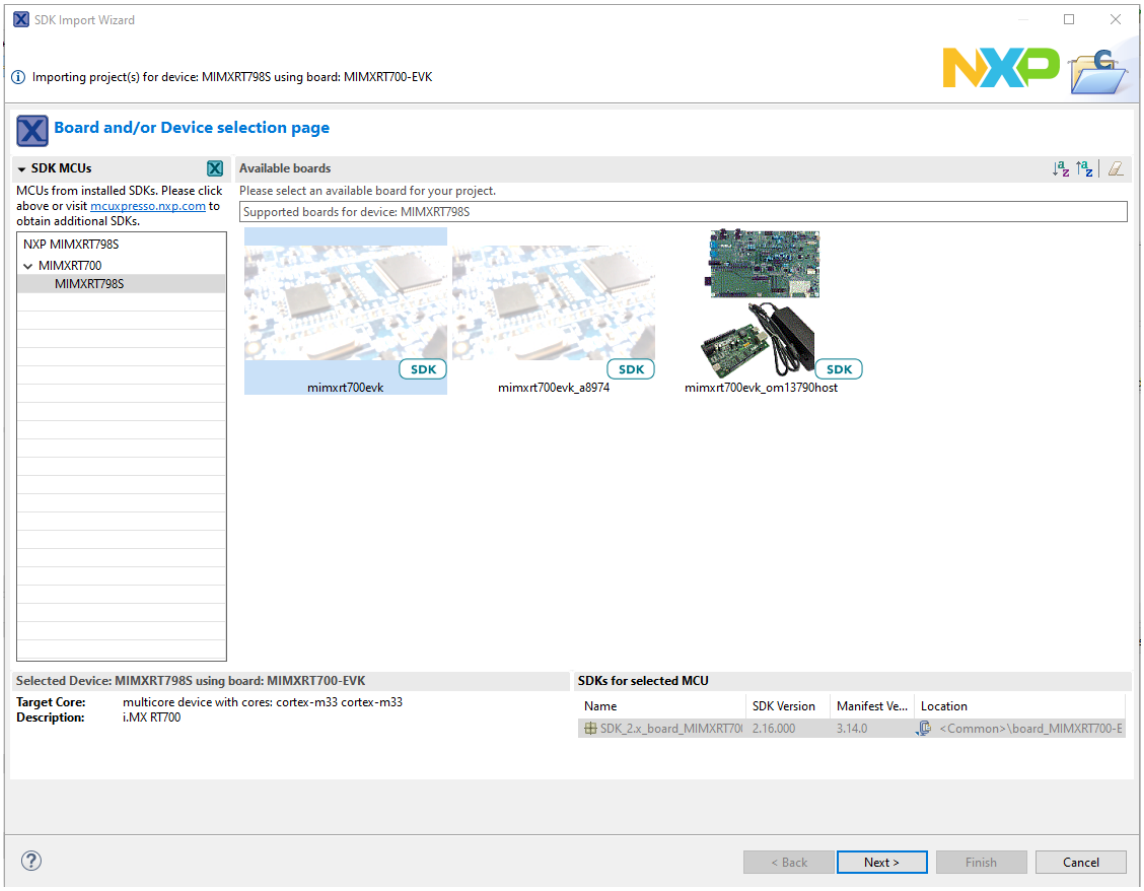
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK.



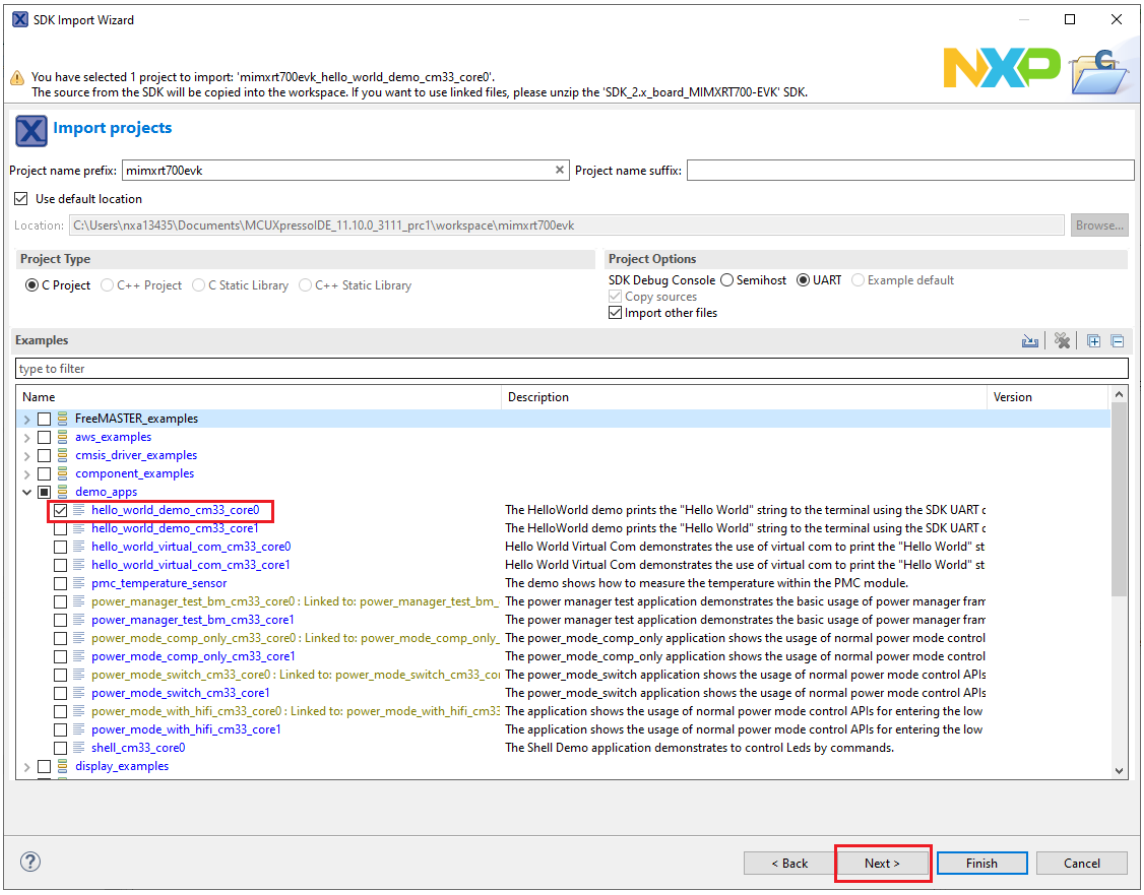
   In the window that appears, click **OK**and wait until the import has finished.

2. On the **Quickstart Panel**, click **Import SDK example(s)**.

3. In the window that appears, select **mimxrt700evk** or **kits** and click **Next**.



4. Expand the *demo_apps* folder and select **hello_world_demo_cm33_core0**. Then, click **Next**.

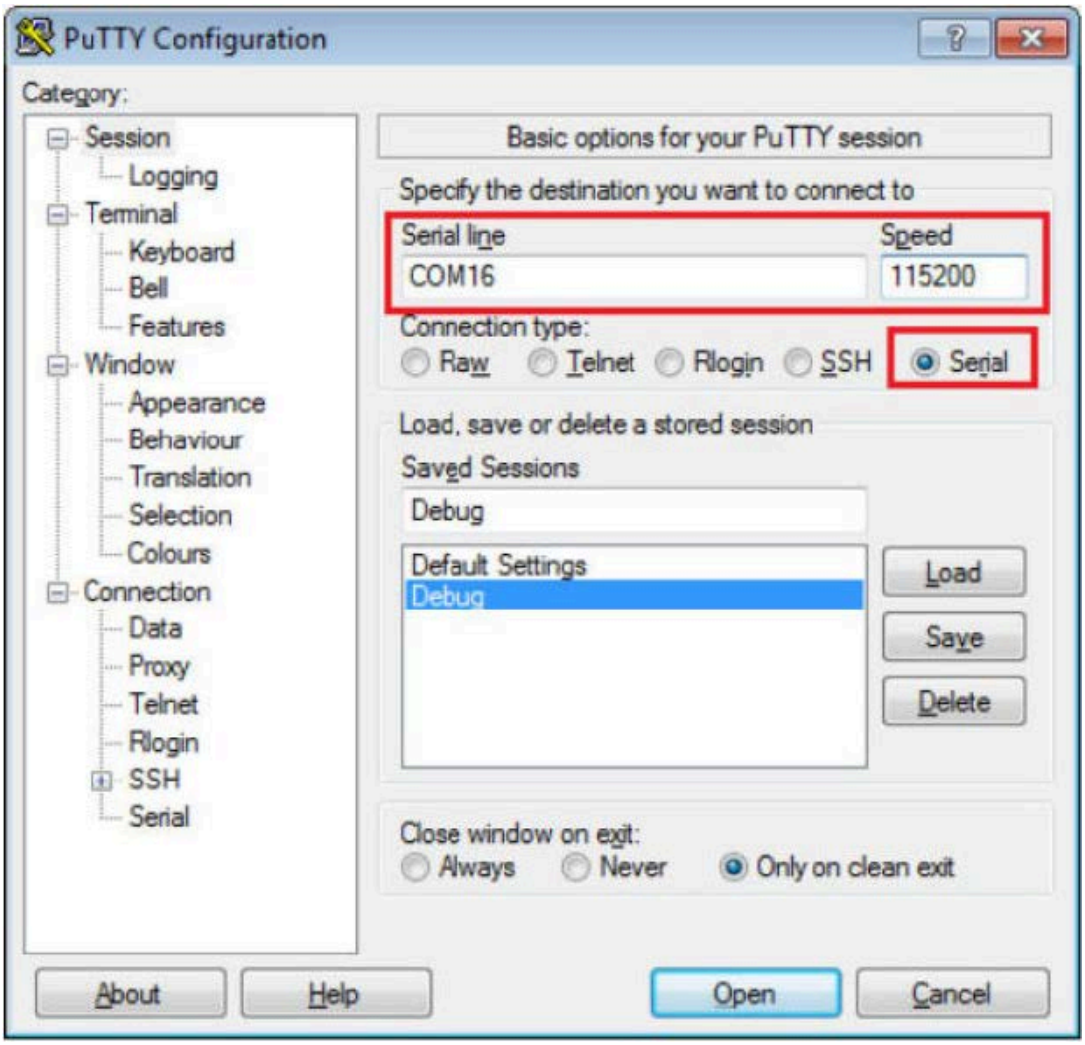


5. Click **Finish**.

Parent topic:*Run a demo using MCUXpresso IDE*

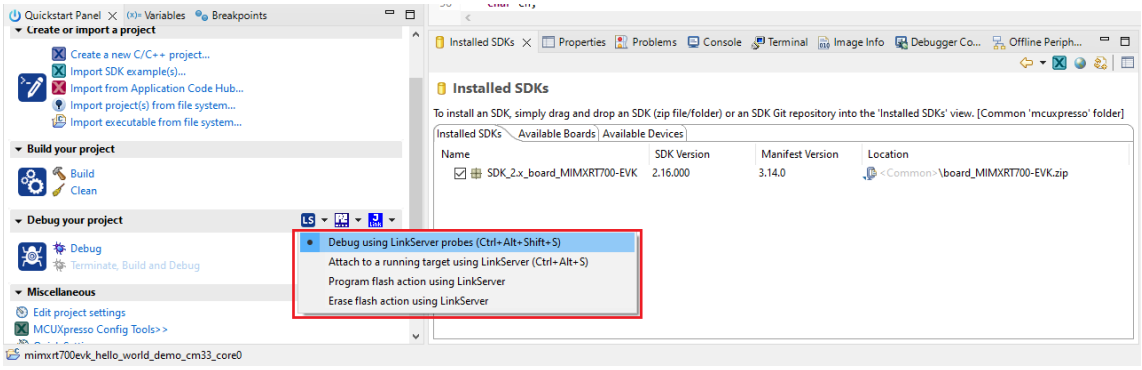
Run an example application To download and run the application, perform the following steps:

1. Determine the debug interface that comes loaded on your specific hardware platform.
   - For MIMXRT700-EVK, MCU-LINK is used.
   - J54 is used as the debugging port which provides both debugging functionality and console out.
2. Connect USB cable between J54 on EVK and PC USB port.
3. Open the terminal application on the PC, such as, PuTTY or TeraTerm, and connect to the debug serial port number. To determine the COM port number, see *How to determine COM port*. Configure the terminal with these settings:
   - 115200 baud rate or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the *board.h* file)
   - No parity
   - 8 data bits

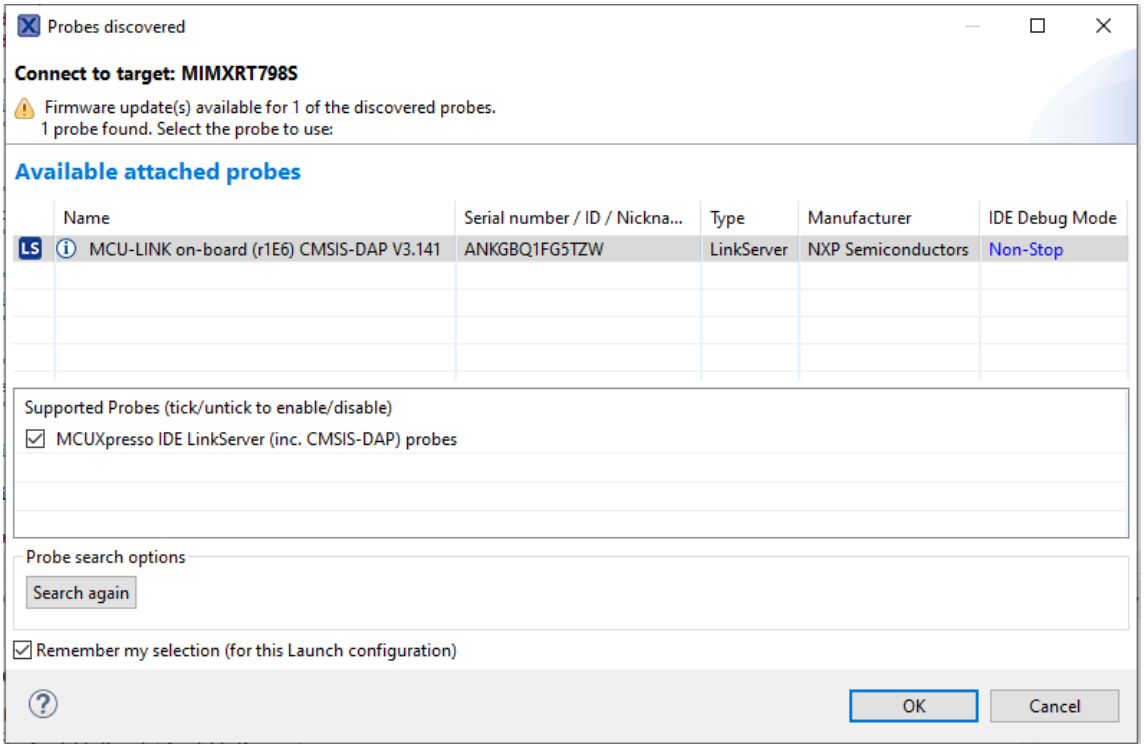


- 1 stop bit

4. On the **Quickstart Panel**, click **Debug** to launch the debug session.

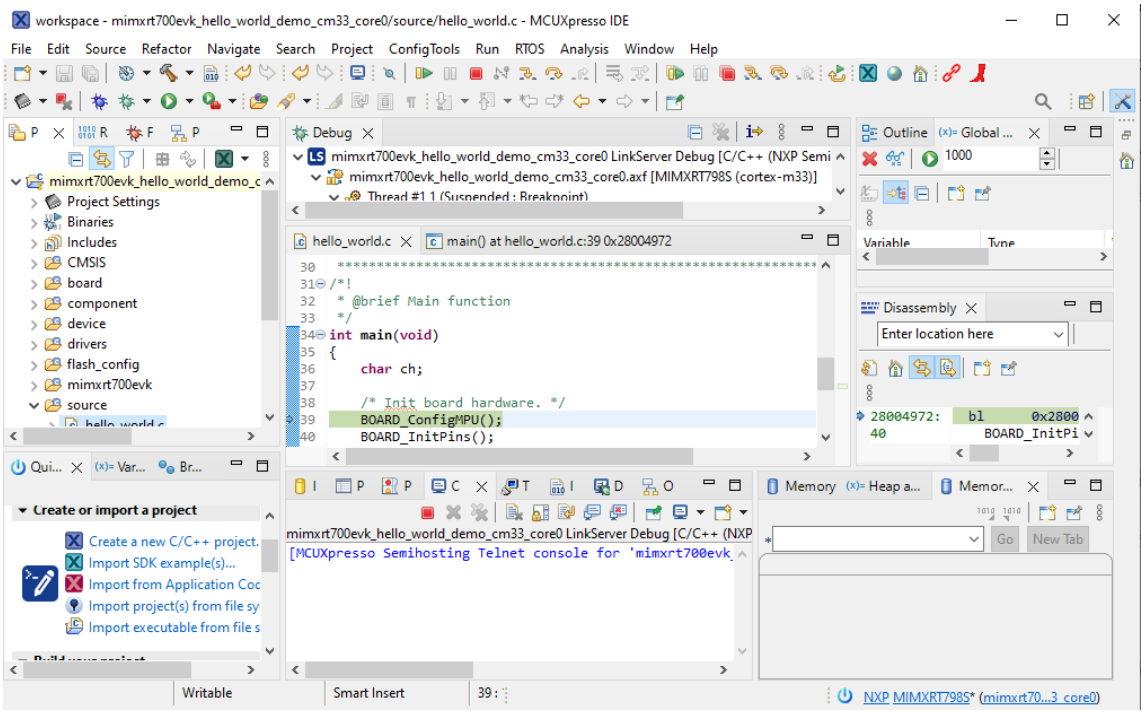


5. The first time you debug a project, the **Debug Emulator Selection** dialog is displayed, showing all supported probes that are attached to your computer. Select the probe through which you want to debug and click **OK**. (For any future debug sessions, the stored probe selection is automatically used, unless the probe cannot be found.)



Note: Make sure that the board is set to XSPI0 flash boot mode.

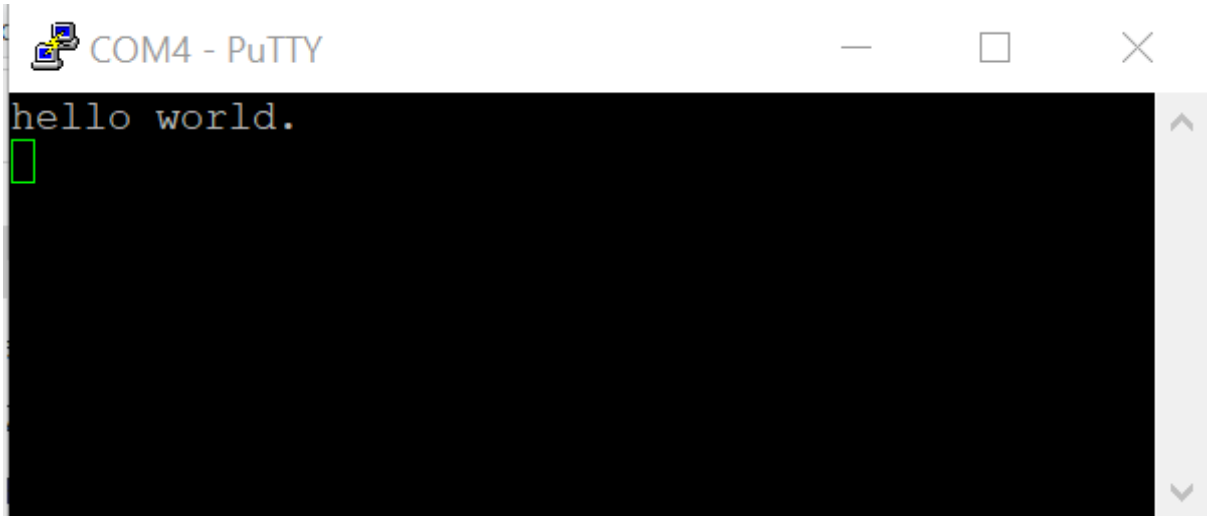
6. The application is downloaded to the target and automatically runs to main().



7. To start the application, click the **Resume**.

The hello_world application is now running and a banner is displayed on the terminal. If not,

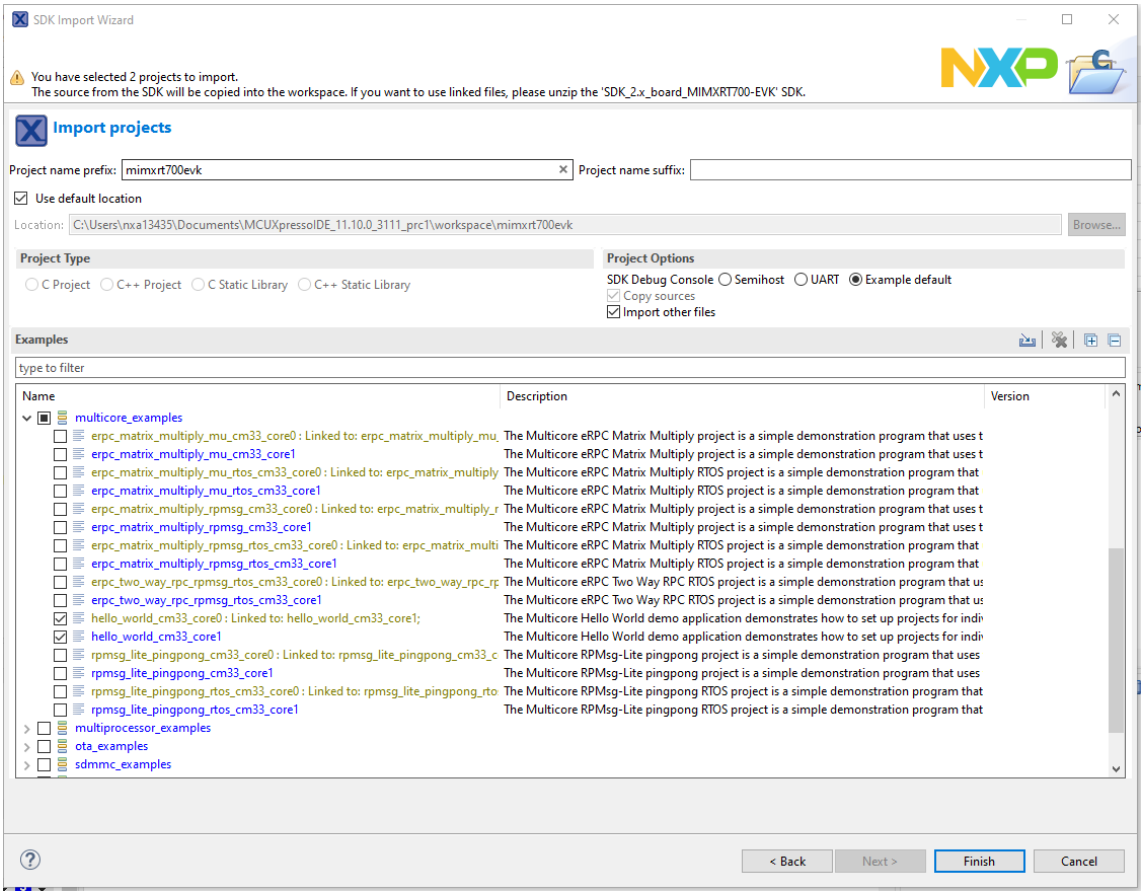
check your terminal settings and connections.



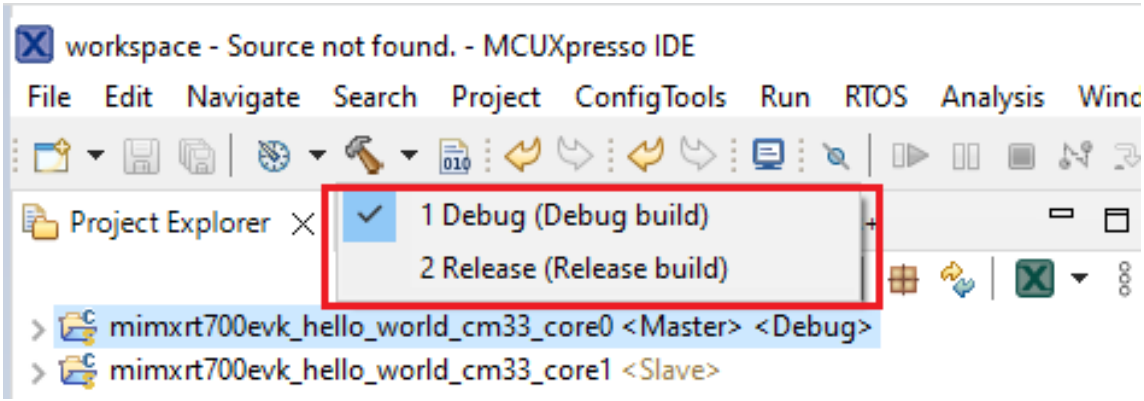
Parent topic:*Run a demo using MCUXpresso IDE*

Build a multicore example application { This section describes the steps required to configure MCUXpresso IDE to build, run, and debug multicore example applications. The following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello_world example application targeted for the **mimxrt700evk** hardware platform is used as an example.

1. Multicore examples are imported into the workspace in a similar way as single core applications, explained in Build an example application. When the SDK zip package is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, select **mimxrt700evk** and click **Next**.
2. Expand the *multicore_examples* folder and select **hello_world_cm33_core0**. The hello_world_cm33_core1 counterpart project is automatically imported with the cm33_core0 project, because the multicore examples are linked together and there is no need to select it explicitly. Click **Finish**.



3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the hello_world_cm33_core0 project (multicore master project) in the Project Explorer. Then choose the appropriate build target, Debug or Release, by clicking the downward facing arrow next to the hammer icon, as shown in *Figure 2*. For this example, select **Debug**.

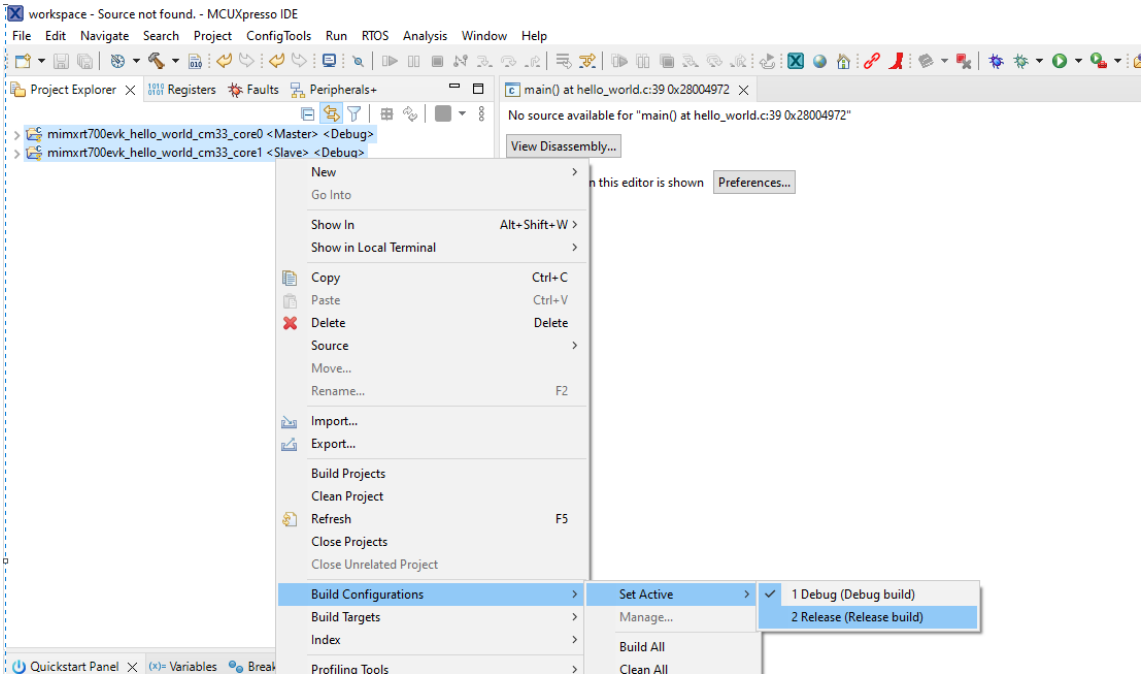


Press the **Build** button to start the multi-core project build.. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm33_core0) also makes the referenced auxiliary core application (cm33_core1) to build.

Note:

When the **Release** build is requested, it is necessary to change the build configuration of both the primary and auxiliary core application projects first. To do this, select both projects in the Project Explorer view and then right click which displays the context-sensitive menu. Select **Build Configurations** -> **Set Active** -> **Release**. This alternate navigation using the menu item is **Project** -> **Build Configuration** -> **Set Active** -> **Release**. After switching to the **Release** build configuration, the build of the multicore example can be started by

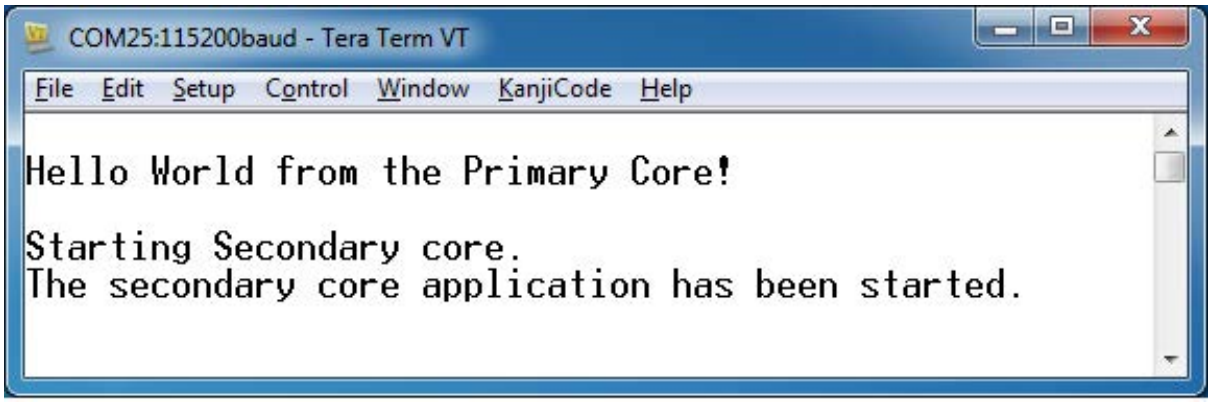
triggering the primary core application (cm33_core0) build.



Parent topic:*Run a demo using MCUXpresso IDE*

Run a multicore example application The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in Run an example application. These steps are common for both single-core applications and the primary side of dual-core applications, ensuring both sides of the multicore application are properly loaded and started. Select the cm33_core1 project and start debugging the cm33_core1 project. Then, select the cm33_core0 project and start debugging the cm33_core0 project.

After clicking **Resume All Debug** sessions, the hello_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



Parent topic:*Run a demo using MCUXpresso IDE*

Run a demo application using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK.

Note:

IAR Embedded Workbench for Arm version 9.50.1 is used in the following example, and the IAR toolchain should correspond to the latest supported version, as described in the *MCUXpresso SDK Release Notes for MIMXRT700-EVK* (document MCUXSDKRT798RN).

When debug or run the flash_debug or flash_release targets, make sure that the board is configured as XSPI0 boot mode (SW10 1 - OFF, 2 - ON).

Build an example application Do the following steps to build the hello_world example application.

1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/iar
```

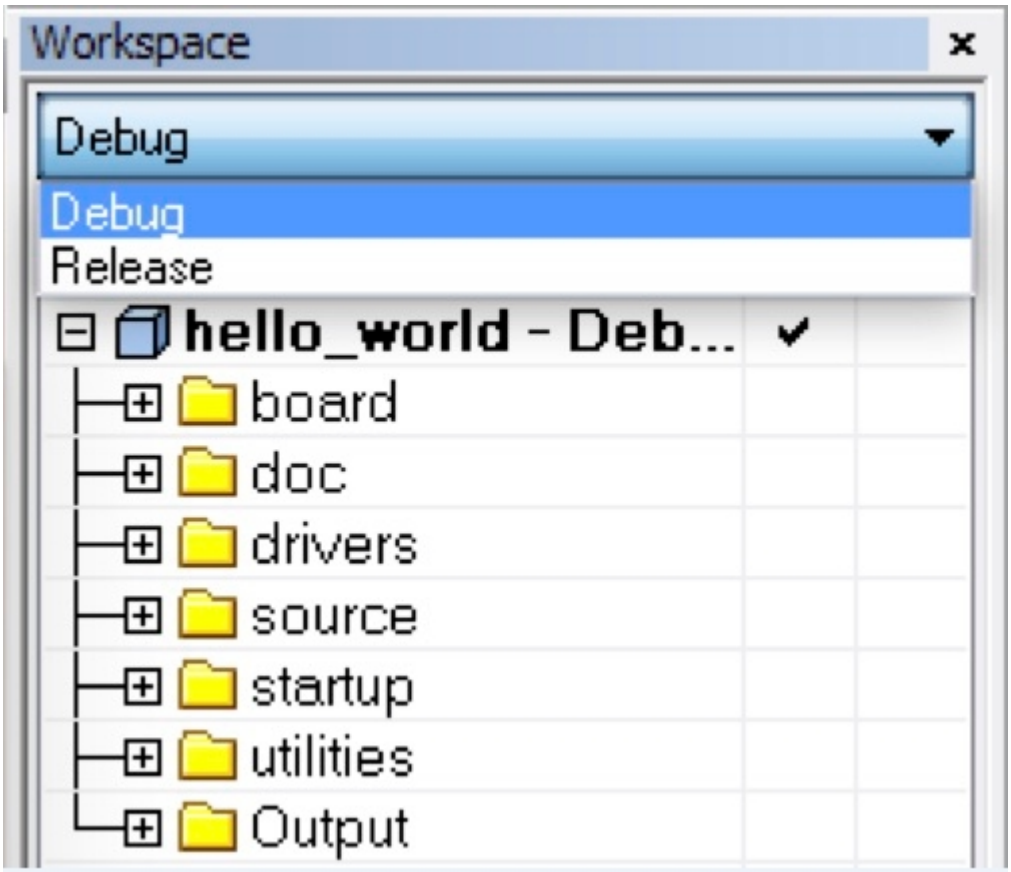
   Using the MIMXRT700-EVK hardware platform as an example, the hello_world workspace is located in:

```
<install_dir>/boards/mimxrt700evk/demo_apps/hello_world/cm33_core0/iar/hello_world_cm33_
↪core0.eww
```

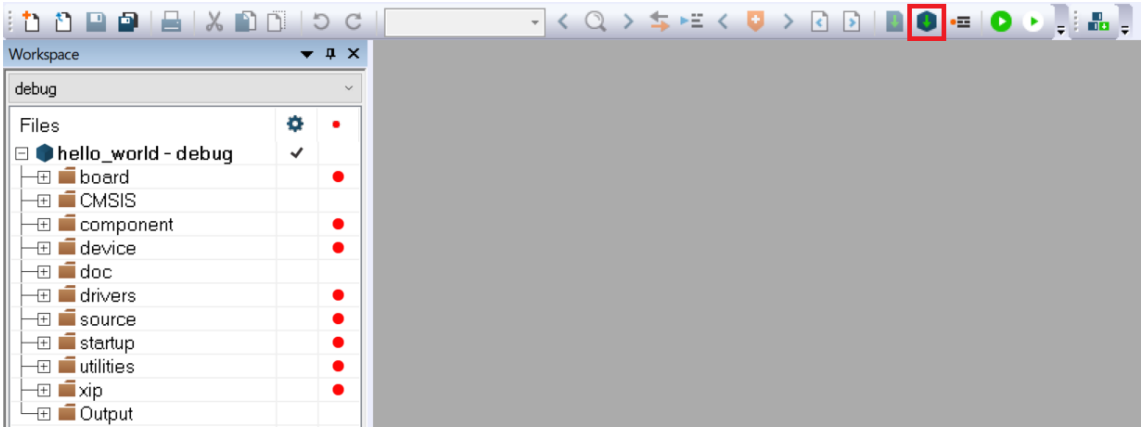
   Other example applications may have additional folders in their path.

2. Select the desired build target from the drop-down menu.

   For this example, select **hello_world** – **debug**.



3. To build the demo application, click **Make**, highlighted in red in Figure 2.

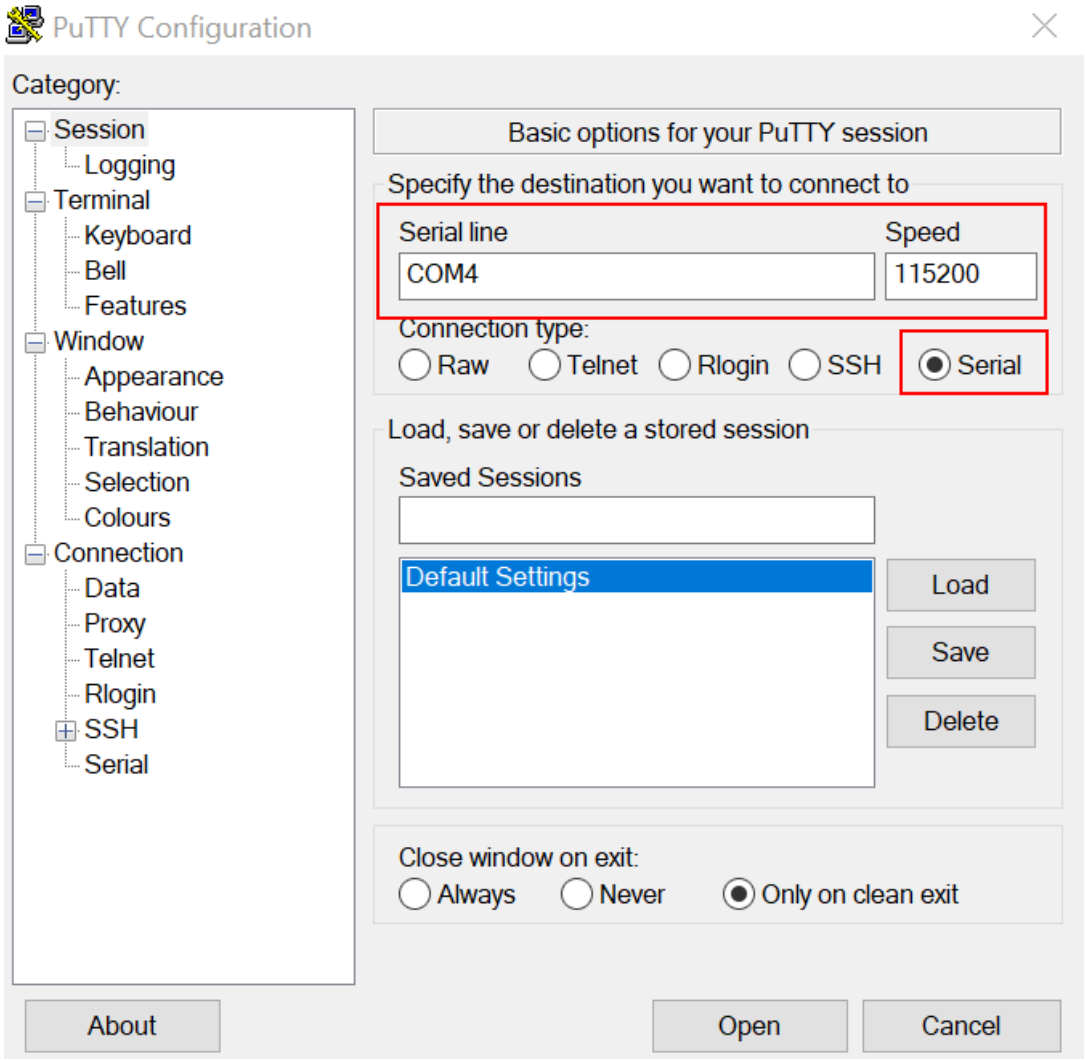


4. The build completes without errors.

Parent topic:*Run a demo application using IAR*

Run an example application To download and run the application, perform these steps:

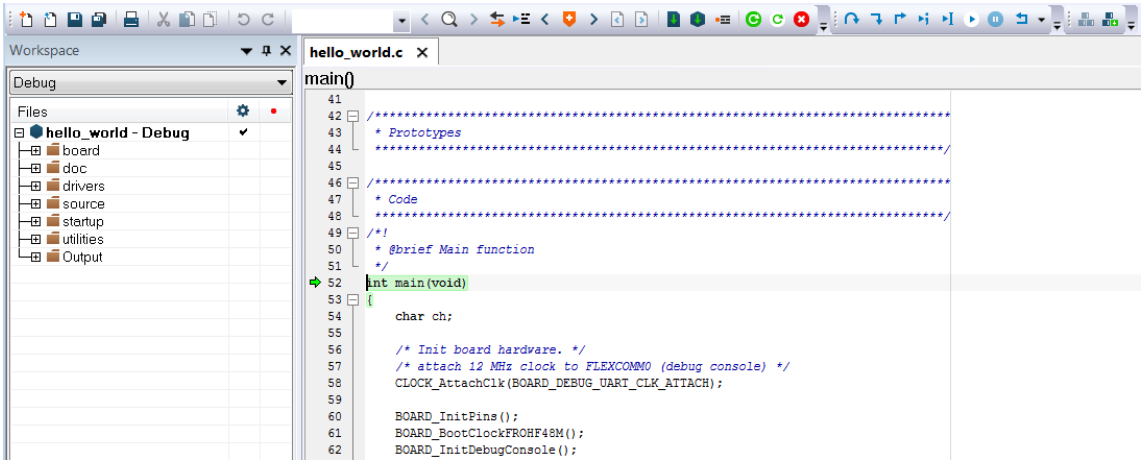
1. To get the MCU-Link firmware ready, see *Updating debugger firmware*.
2. Connect the development platform to your PC via USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see *How to determine COM port*). Configure the terminal with these settings:
    1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the *board.h* file)
    2. No parity
    3. 8 data bits



4. 1 stop bit

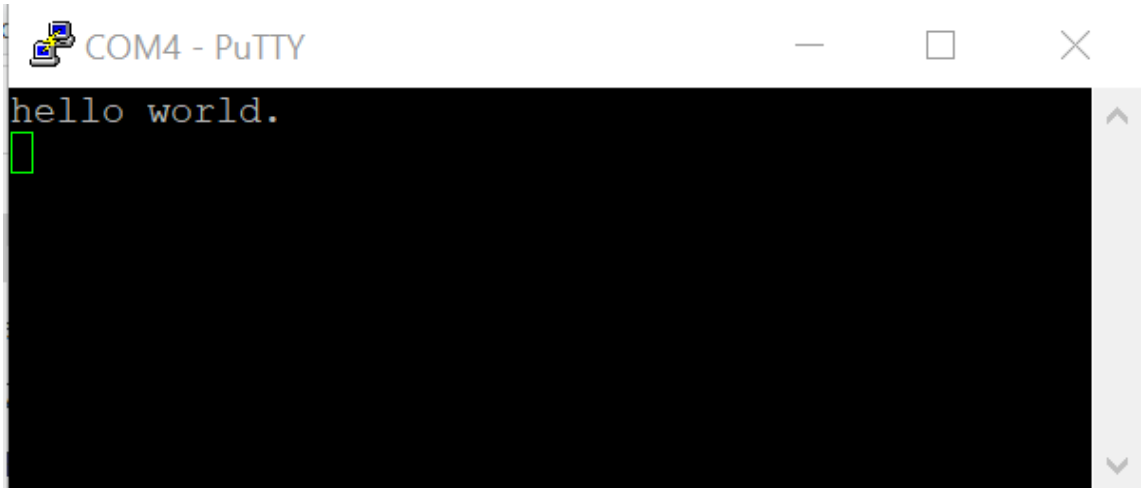
4. In IAR, click the **Download and Debug** button to download the application to the target.

5. The application is then downloaded to the target and automatically runs to the main() function.



6. Run the code by clicking the **Go** button.

7. The hello_world application is now running and a banner is displayed on the terminal. If it does not appear, check your terminal settings and connections.



Parent topic:*Run a demo application using IAR*

Build and run a multicore example application This section describes the steps to build and run a multicore application. The demo applications workspace files are available in the folder:

<install_dir>/boards/<board_name>/multicore_examples/<application_name>/<core_id>/iar

Begin with a simple dual-core version of the Hello World application. The multicore Hello World IAR workspaces are available in the folder:

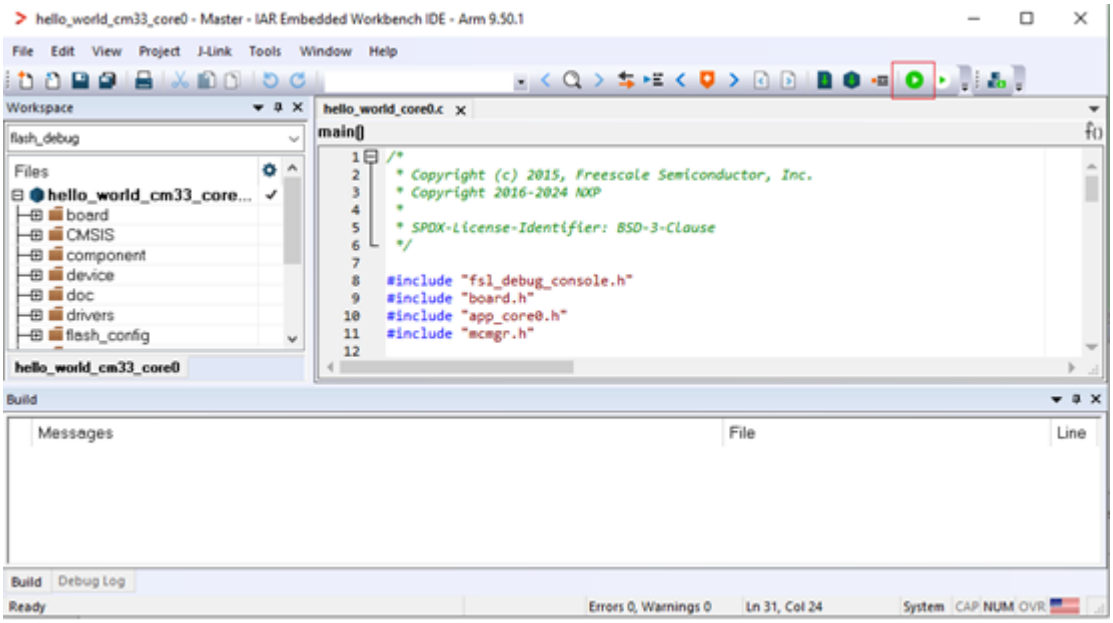
<install_dir>/boards/mimxrt700evk/multicore_examples/hello_world/cm33_core0/iar/hello_world_cm33_core0.ew

<install_dir>/boards/mimxrt700evk/multicore_examples/hello_world/cm33_core1/iar/hello_world_cm33_core1.ew

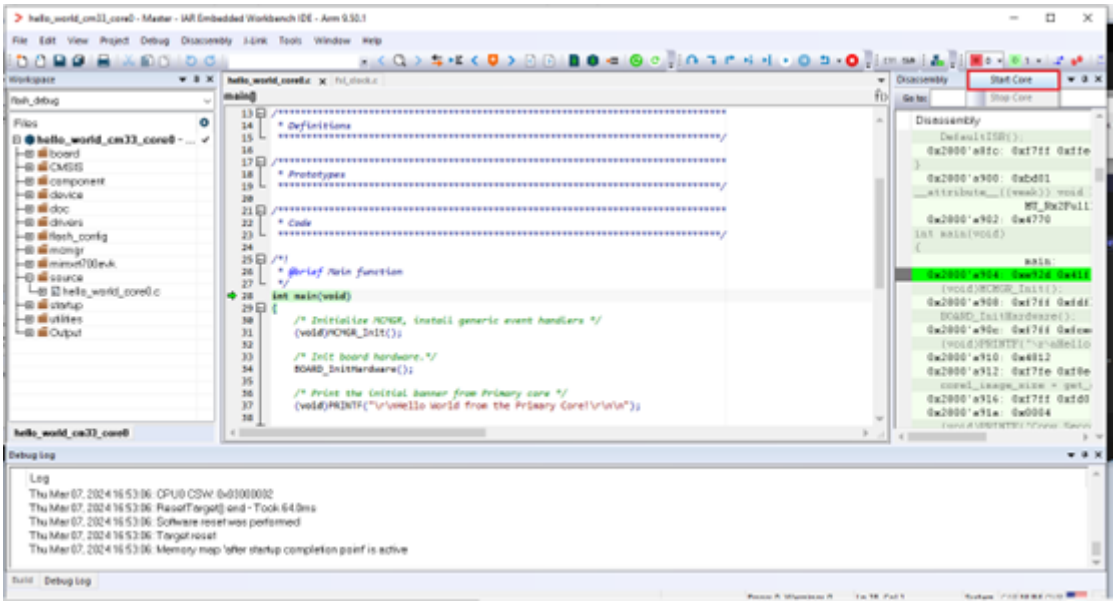
Build both applications separately by clicking the **Make** button. Build the application for the auxiliary core (cm33_core1) first, because the primary core application project (cm33_core0) must know the auxiliary core application binary when running the linker. When the auxiliary core application binary is not ready, it is impossible to finish the primary core linker.

By default, the primary core flash_debug target links the auxiliary core debug target, and the primary core flash_release target links the auxiliary core release target. During the primary core execution, the auxiliary core image is copied from flash into the SRAM and executed.

1. Build the CM33_core1 and CM33_core0 projects respectively.
2. Only click the **Download** and **Debug** button on the CM33_core0 project, IAR could help start to debug a multicore project. It is user-friendly to debug multicore examples with CMSIS-DAP on IAR (Multicore Project is set on **Debugger** → **Multicore** window).



3. Start core0 and then start core1 on the **CM33_core0** project.



4. The Hello_World multicore demos are now running. A banner appears on the terminal and the LED blinks. If this is not true, check your terminal settings and connections.

```
Hello World from the Primary Core!

Copy Secondary core image to address: 0x20600000, size: 12272
Starting Secondary core.
The secondary core application has been started.
```

Parent topic:*Run a demo application using IAR*

Run a demo using Keil MDK/µVision

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK.

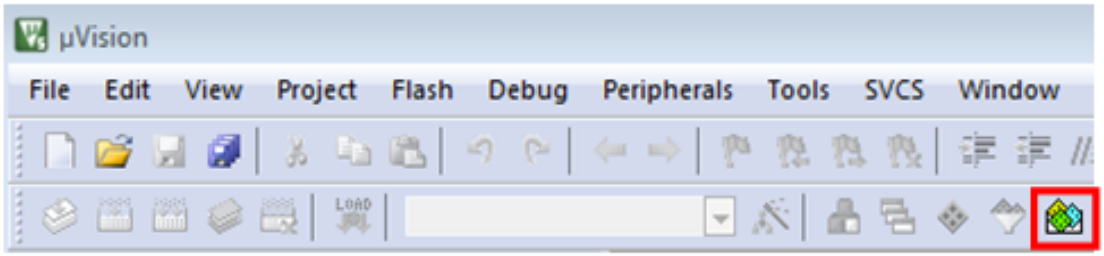
The hello_world demo application is used as an example, although these steps can be applied to any demo or example application in the MCUXpresso SDK.

Note:

Make sure that the board is configured as XSPI0 boot mode when debugging or running the examples.

Install CMSIS device pack After the MDK tools are installed, Cortex Microcontroller Software Interface Standard (CMSIS) device packs must be installed to fully support the device from a debug perspective. These packs include things such as memory map information, register definitions and flash programming algorithms. Follow these steps to install the appropriate CMSIS pack.

1. Open the MDK IDE, which is called µVision. In the IDE, select the **Pack Installer** icon.



2. After the installation finishes, close the Pack Installer window and return to the µVision IDE.

Parent topic:*Run a demo using Keil MDK/µVision*

Build an example application To build an example application, perform the following steps:

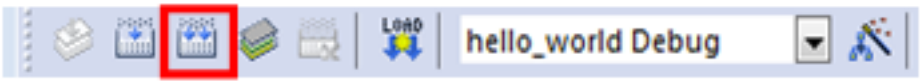
1. Open the desired example application workspace in:

   <install_dir>/boards/<board_name>/<example_type>/<application_name>/<core_id>/mdk
   The workspace file is named as <demo_name>.uvmpw.

   For this specific example, the actual path is:

   <install_dir>/boards/mimxrt700evk/demo_apps/hello_world/cm33_core0/mdk/hello_world_demo_cm33_core

2. To build the demo project, select **Rebuild**, as shown in *Figure 1*, highlighted in red.



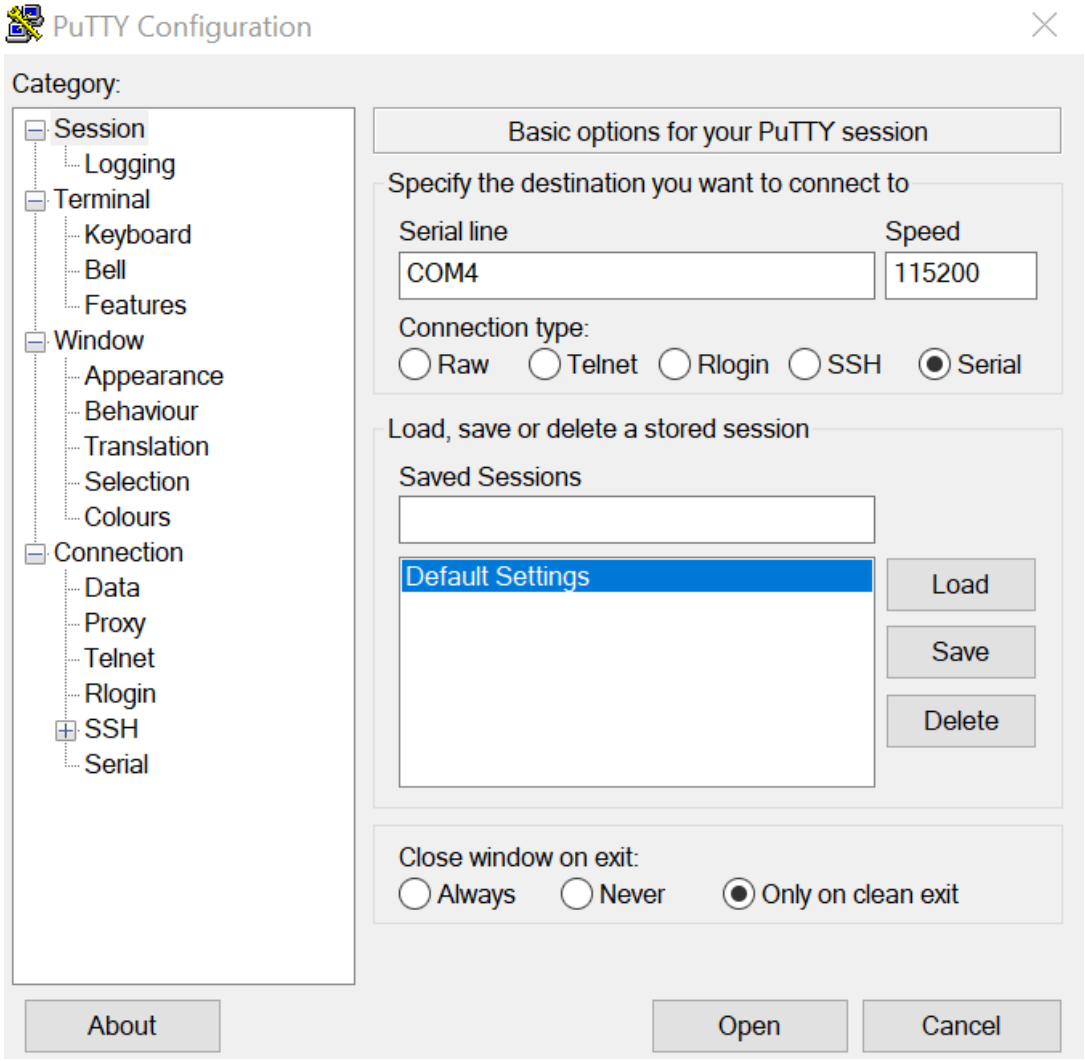
3. The build completes without errors.

Parent topic:*Run a demo using Keil MDK/µVision*

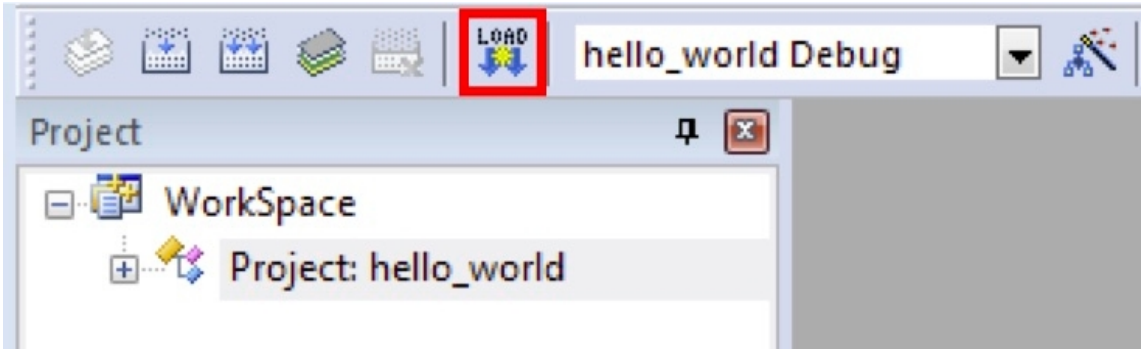
Run an example application To download and run the application, perform these steps:

1. Determine the debug interface that comes loaded on your specific hardware platform.
   - For boards with the CMSIS-DAP/mbed/DAPLink interface, visit Windows serial configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux operating system, this step is not required.
   - If using J-Link either a standalone debug pod or MCU-LINK, install the J-Link software (drivers and utilities) from Segger.
2. Connect the development platform to your PC via USB cable using MCU-LINK USB connector.

3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see *How to determine COM port*). Configure the terminal with these settings:
   - 115200 or 9600 baud rate, depending on your board (reference the BOARD_DEBUG_UART_BAUDRATE variable in the *board.h* file)
   - No parity
   - 8 data bits
   - 1 stop bit

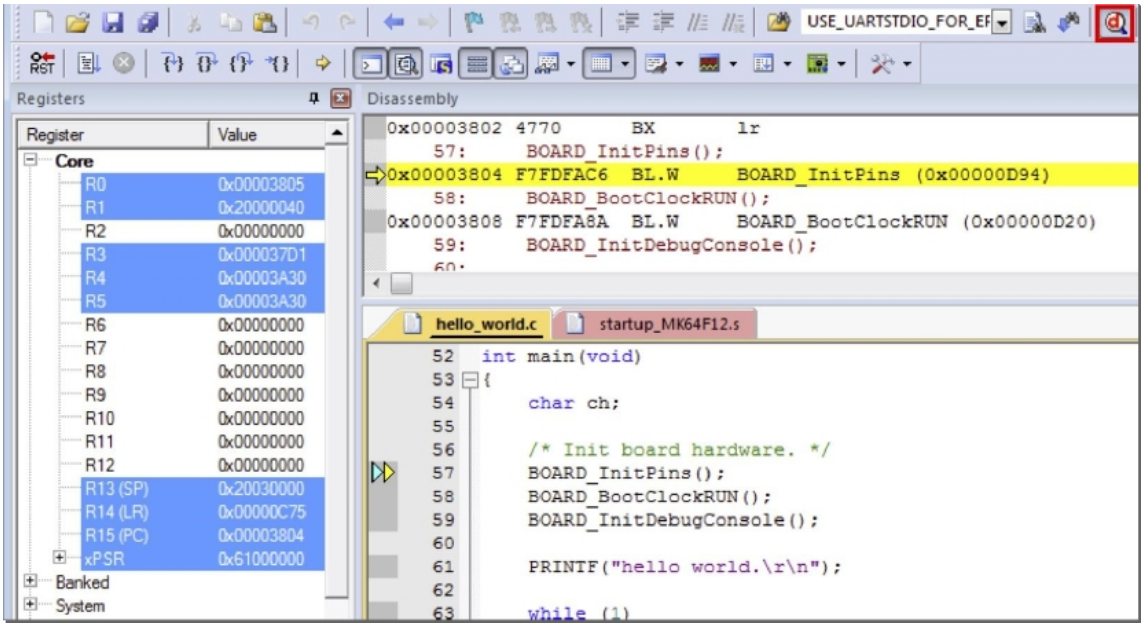


4. In µVision, after the application is built, click the **Download** button to download the application to the target.

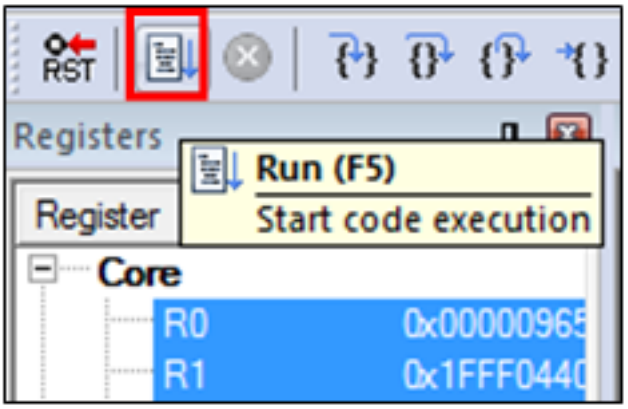


5. After clicking the **Download** button, the application downloads to the target and is running.

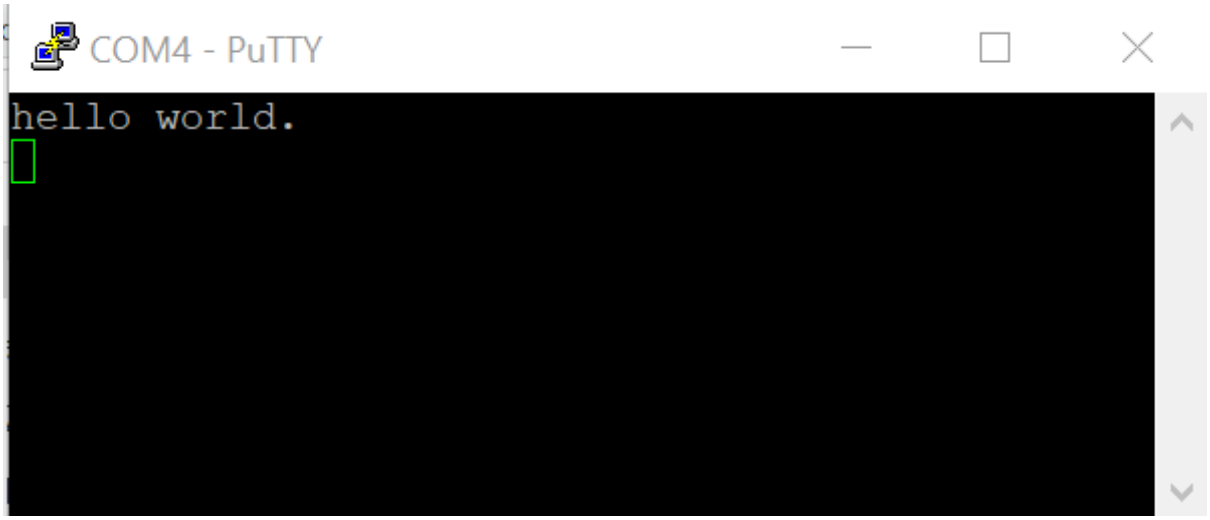
To debug the application, click the **Start/Stop Debug Session** button, highlighted in red in *Figure 3*.



6. Run the code by clicking the **Run** button to start the application.



The hello_world application is now running and a banner is displayed on the terminal. If this does not appear, check your terminal settings and connections.



Parent topic:*Run a demo using Keil MDK/µVision*

Build a multicore example application This section describes the particular steps that need to be done in order to build and run a multicore application.

The multicore Hello World MDK workspaces are located in this folder:

<install_dir>/boards/mimxrt700evk/multicore_examples/hello_world/cm33_core0/mdk/hello_world_cm33_core0.u

<install_dir>/boards/mimxrt700evk/multicore_examples/hello_world/cm33_core1/mdk/hello_world_cm33_core1.u

To build both applications separately, click the **Rebuild** button. Build the application for the auxiliary core (cm33_core1) first, because the primary core application project (cm33_core0) must know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

Parent topic:*Run a demo using Keil MDK/µVision*

Run a multicore example application This section describes the steps to run a multicore example application. The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory.

1. To download and run the multicore application, switch to the primary core application project and perform Step 1 – Step 3, as described in Run an example application.
2. Run the primary core project and then run the secondary core project.
3. Hello_World multicore demos are now running. A banner appears on the terminal and the LED blinks. If this is not true, check your terminal settings and connections.

```
Hello World from the Primary Core!

Copy Secondary core image to address: 0x20600000, size: 12844
Starting Secondary core.
The secondary core application has been started.
```

Parent topic:*Run a demo using Keil MDK/µVision*

Run a demo using ARMGCC / VSCODE

This section describes the steps to run an example application from the SDK archive using the ARMGCC / VSCODE toolchain.

Refer to the *running a demo using MCUXpresso VSC* section for detailed instructions on setting up and configuring your project in Visual Studio Code.

Refer to the *CLI* section for detailed instructions on building and running your project from the command line.

Build and run the ezhv_examples

This section describes the steps required to configure the environment to build, run, and debug ezhv example applications. The *<install_dir>/boards/<board_name>/ezhv_examples/led_blinky* is used as an example.

For each example, generally there are two folders, one for the cm33_core0 core and the other for the EZHV core. For the led_blinky example, they are:

<install_dir>/boards/<board_name>/ezhv_examples/led_blinky/cm33_core0

<install_dir>/boards/<board_name>/ezhv_examples/led_blinky/ezhv

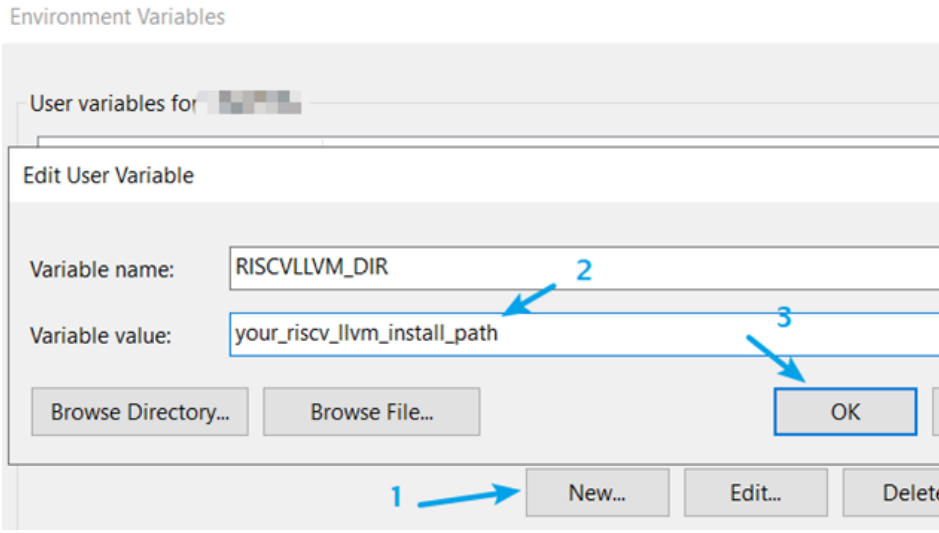
Build the application for the auxiliary core (ezhv) first, because the primary core application project (cm33_core0) must know the auxiliary core application binary when running the linker. When the auxiliary core application binary is not ready, it is impossible to finish the primary core linker. This section focuses on the auxiliary core (ezhv). For details about building the primary core for different toolchain, see the corresponding chapter.

Set the toolchain for EZHV To set the toolchain for EZHV, perform the following steps:

1. Download and install the RISCV LLVM toolchain.
2. Add a system environment variable RISCVLLVM_DIR, and set the value to the RISCV LLVM install path.
   - For Linux or Mac, run the command:

     ```
     export RISCVLLVM_DIR=<your_risc_llvm_install_path>
     ```

   - For Windows, as shown in Set the toolchain for EZHV, perform the following steps:
     1. Click **Windows Start** -> **Settings**.
     2. Search and open **Edit environment variable for your account**
3. Create a new variable.



3. Download Ninja from this link. Add the path ninja executable to system environmental variable **PATH**. When it finishes, run the ninja – version in command line to verify the installation. The result looks like:

   ```
   $ ninja --version
   1.10.1
   ```

4. Download and install CMake.

Parent topic:*Build and run the ezhv_examples*

Build the EZHV project To build the EZHV project, go to the project folder *<install_dir>/boards/<board_name>/ezhv_examples/<example>/ezhv/riscvllvm*.

- For Windows, click the **build_debug.bat**.
- For Linux or Mac, run the build_debug.sh in the command line.

When the build finishes, a binary *ezhv_image.bin* is generated in the folder *<install_dir>/boards/<board_name>/ezhv_examples/<example>/ezhv/binary*.

Parent topic:*Build and run the ezhv_examples*

Run the project There are two ways to run the project: Run with primary core, or run independently.

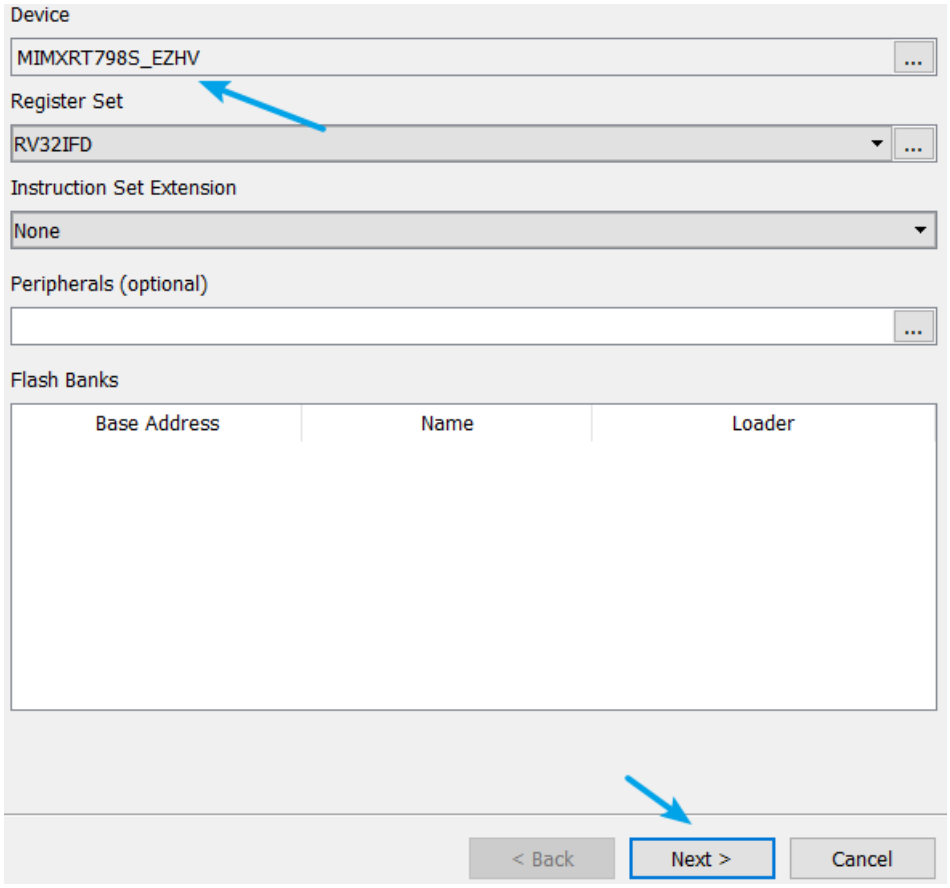
- Run with primary core.

  After EZHV binary *ezhv_image.bin* is created, follow the primary core project guide to build the primary core project and run.

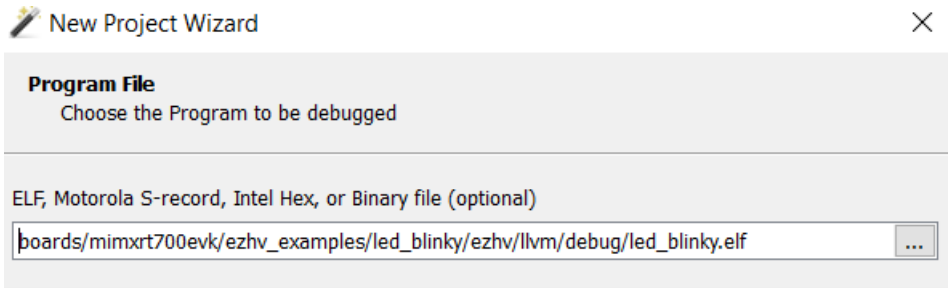
- Debug the EZHV independently.

  Currently SEGGER Ozone can be used to run and debug the EZHV executable.

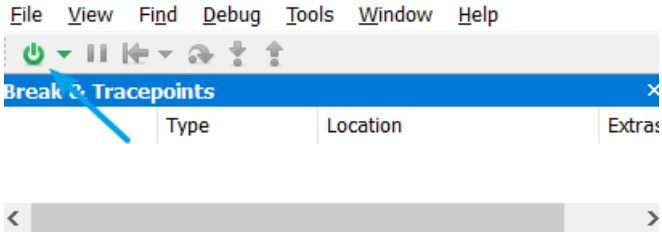
  1. An external J-Link plus is needed, connect JP18 1-2 on the mimxrt700evk board to use external debugger and connect external J-Link to J18.
  2. Download and install the J-Link.
  3. Download and install the SEGGER Ozone.
  4. Open Ozone and create project, click **File** -> **New** -> **New Project Wizard**.
  5. Select the device **MIMXRT798S_EZHV**.



  6. Select the built EZHV executable.



  7. On the Ozone window, click the button to run.



Parent topic:*Build and run the ezhv_examples*

How to determine COM port

This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform.

1. **Linux**: The serial port can be determined by running the following command after the USB Serial is connected to the host:

   ```
   $ dmesg | grep "ttyUSB"
      [503175.307873] usb 3-12: cp210x converter now attached to ttyUSB0
      [503175.309372] usb 3-12: cp210x converter now attached to ttyUSB1
   ```

   There are two ports: one is for Compute Domain and the other is for Sense Domain.

2. **Windows**: To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.
3. In the Device Manager, expand the **Ports (COM & LPT)** section to view the available ports. The COM port names are different for all the NXP boards.

How to define IRQ handler in CPP files

With MCUXpresso SDK, users could define their own IRQ handler in application level to override the default IRQ handler. For example, to override the default PIT_IRQHandler define in startup_DEVICE.s, application code like *app.c* can be implement like:

```
c
void PIT_IRQHandler(void)
{
    // Your code
}
```

When application file is CPP file, like *app.cpp*, then extern "C" should be used to ensure the function prototype alignment.

```
cpp
extern "C" {
    void PIT_IRQHandler(void);
}
void PIT_IRQHandler(void)
{
    // Your code
}
```

Updating debugger firmware

Updating MCU-Link board firmware MCU-Link is the NXP single, unified debug probe architecture for all NXP general-purpose Arm Cortex-M based MCUs. MCU-Link is implemented on the

latest evaluation boards (known as MCU-Link OB) of NXP. This firmware in this debug interface may be updated using the host computer utility called MCU-Link firmware. This typically used when switching between the default debugger protocol (CMSIS-DAP) to SEGGER J-Link, or for updating this firmware with new releases of these. This section contains the steps to re-program the debug probe firmware.

NXP provides the MCU-Link firmware, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto MCU-Link. The utility can be downloaded from MCU-Link Debug Probe Architecture.

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux operating system, follow the similar steps.

1. Install the MCU-Link firmware.
2. Unplug the USB cable of the board.
3. Enable ISP mode on MCU-Link Core - LPC55xxx (install the jumper labeled ISP Enable).
4. Connect the probe to the host via USB (use Link USB connector).
5. Open a command shell and call the appropriate script located in the MCU-Link firmware installation directory (<*firmware install dir*>).
    1. To program CMSIS-DAP debug firmware: <*firmware install dir*>*/scripts/program_CMSIS*
    2. To program J-Link debug firmware: <*LPCScrypt install dir*>*/scripts/program_JLINK*
6. Disable ISP mode on MCU-Link Core - LPC55xxx (remove the jumper installed in Step 3).
7. Repower the board by removing the USB cable and plugging it in again.

Parent topic:*Updating debugger firmware*

Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.4 Getting Started with MCUXpresso SDK GitHub

1.4.1 Getting Started with MCUXpresso SDK Repository

Welcome to the **GitHub Repository SDK Guide**. This documentation provides instructions for setting up and working with the MCUXpresso SDK distributed in a **multi-repository model**. The SDK is distributed across multiple GitHub repositories and managed using the **Zephyr West** tool, enabling modular development and streamlined workflows.

Overview

The GitHub Repository SDK approach offers:

- **Modular Structure**: Multiple repositories for flexibility and scalability.
- **Zephyr West Integration**: Simplified repository management and synchronization.
- **Cross-Platform Support**: Designed for MCUXpresso SDK development environments.

Benefits of the Multi-Repository Approach

- **Scalability**: Easily add or update components without impacting the entire SDK.
- **Collaboration**: Enables distributed development across teams and repositories.
- **Version Control**: Independent versioning for components ensures better stability.
- **Automation**: Zephyr West simplifies dependency handling and repository synchronization.

Setup and Configuration

Follow these steps to prepare your development environment:

Development Tools Installation This guide explains how to install the essential tools for development with the MCUXpresso SDK.

Quick Start: Automated Installation (Recommended)

The **MCUXpresso Installer** is the fastest way to get started. It automatically installs all the basic tools you need.

1. **Download the MCUXpresso Installer** from: MCUXpresso Installer
2. **Run the installer** and select **"MCUXpresso SDK Developer"** from the menu
3. **Click Install** and let it handle everything automatically

Manual Installation

If you prefer to install tools manually or need specific versions, follow these steps:

Essential Tools

Git - Version Control

What it does: Manages code versions and downloads SDK repositories from GitHub.

Installation:

- Visit git-scm.com
- Download for your operating system
- Run installer with default settings
- **Important**: Make sure "Add Git to PATH" is selected during installation

Setup:

```
git config --global user.name "Your Name"
git config --global user.email "youremail@example.com"
```

Python - Scripting Environment

What it does: Runs build scripts and SDK tools.

Installation:

- Install Python **3.10 or newer** from python.org
- **Important**: Check "Add Python to PATH" during installation

West - SDK Management Tool

What it does: Manages SDK repositories and provides build commands. The west tool is developed by the Zephyr project for managing multiple repositories.

Installation:

```
pip install -U west
```

Minimum version: 1.2.0 or newer

Build System Tools

CMake - Build Configuration

What it does: Configures how your projects are built.

Recommended version: 3.30.0 or newer

Installation:

- **Windows**: Download .msi installer from cmake.org/download
- **Linux**: Use package manager or download from cmake.org
- **macOS**: Use Homebrew (brew install cmake) or download from cmake.org

Ninja - Fast Build System

What it does: Compiles your code quickly.

Minimum version: 1.12.1 or newer

Installation:

- **Windows**: Usually included, or download from ninja-build.org
- **Linux**: sudo apt install ninja-build or download binary
- **macOS**: brew install ninja or download binary

Ruby - IDE Project Generation (Optional)

What it does: Generates project files for IDEs like IAR and Keil.

When needed: Only if you want to use traditional IDEs instead of VS Code.

Installation: Follow the Ruby environment setup guide

Compiler Toolchains

Choose and install the compiler toolchain you want to use:

Toolchain	Best For	Download Link	Environment Variable
Arm GCC (Recommended)	Most users, free	Arm GNU Toolchain	ARMGCC_DIR
IAR EWARM	Professional development	IAR Systems	IAR_DIR
Keil MDK	Arm ecosystem	Arm Developer	MDK_DIR
Arm Compiler	Advanced optimization	Arm Developer	ARMCLANG_DIR
Code Warrior	DSC platforms	NXP CodeWarrior	CW_DIR
Xtensa	DSP core	Tensilica Tools	XCC_DIR

Setting Up Environment Variables

After toolchain installation, set an environment variable so the build system locates it:

Windows:

```
# Example for ARM GCC installed in C:\armgcc
setx ARMGCC_DIR "C:\armgcc"
```

Linux/macOS:

```
# Add to ~/.bashrc or ~/.zshrc
export ARMGCC_DIR="/usr"  # or your installation path
```

Verify Your Installation

After installation, verify everything works by opening a terminal/command prompt and running these commands:

```
# Check each tool - you should see version numbers
git --version
python --version
west --version
cmake --version
ninja --version
arm-none-eabi-gcc --version  # (if using ARM GCC)
```

Troubleshooting Installation Issues

"Command not found" errors:

- The tool isn't in your system PATH
- **Solution**: Add the installation directory to your PATH environment variable

Python/pip issues:

- Try using `python3` and `pip3` instead of `python` and `pip`

- On Windows, run the Command Prompt as an Administrator

Slow downloads:

- Add timeout option: pip install -U west --default-timeout=1000
- Use alternative mirror: pip install -U west -i https://pypi.tuna.tsinghua.edu.cn/simple

GitHub Repository Setup This guide explains how to initialize your MCUXpresso SDK workspace from GitHub repositories using the west tool. The GitHub Repository SDK uses multiple repositories hosted on GitHub to provide modular, flexible development.

Prerequisites

Verify the requirements:

System Requirements:

- Python 3.8 or later
- Git 2.25 or later
- CMake 3.20 or later
- Build tools for your target platform

Verification Commands:

```
python --version    # Should show 3.8+
git --version       # Should show 2.25+
cmake --version     # Should show 3.20+
west --version      # Should show west tool installation
```

Workspace Initialization

The GitHub Repository SDK uses the Zephyr west tool to manage multiple repositories containing different SDK components.

Step 1: Initialize Workspace

Create and initialize your SDK workspace from GitHub:

Get the latest SDK from main branch:

```
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests.git mcuxpresso-sdk
```

Get SDK at specific revision:

```
west init -m https://github.com/nxp-mcuxpresso/mcuxsdk-manifests.git mcuxpresso-sdk --mr {revision}
```

Note: Replace {revision} *with the desired release tag, such as* v25.09.00

Step 2: Choose Your Repository Update Strategy

Navigate to the SDK workspace:

```
cd mcuxpresso-sdk
```

The west tool manages multiple GitHub repositories containing different SDK components. You have two options for downloading:

Option A: Download All Repositories (Complete SDK)

Download all SDK repositories for comprehensive development:

```
west update
```

This command downloads all the repositories defined in the manifest from GitHub. Initial download takes several minutes and requires ~7 GB of disk space.

Best for:

- Exploring the complete SDK
- Multi-board development projects
- Comprehensive middleware evaluation

Option B: Targeted Repository Download (Recommended)

Download only repositories needed for your specific board or device to save time and disk space:

```
# For specific board development
west update_board --set board your_board_name

# For specific device family development
west update_board --set device your_device_name

# List available repositories before downloading
west update_board --set board your_board_name --list-repo
```

Best for:

- Single board development
- Faster setup and reduced disk usage
- Focused development workflows

Examples:

```
# Update only repositories for FRDM-MCXW23 board
west update_board --set board frdmmcxw23

# Update only repositories for MCXW23 device family
west update_board --set device mcxw23
```

Step 3: Verify Installation

Confirm successful setup:

```
# Verify workspace structure
ls -la
# Should show: manifests/ and mcuxsdk/ directories

# Test build system
west list_project -p examples/demo_apps/hello_world
# Should display available build configurations
```

Advanced Repository Management

The west extension command `update_board` provides advanced repository management capabilities for optimized workspace setup with GitHub repositories.

Board-Specific Setup

Update only repositories required for a specific board:

```
# Update only repositories for specific board, e.g., frdmmcxw23
west update_board --set board frdmmcxw23

# List available repositories for the board before updating
west update_board --set board frdmmcxw23 --list-repo
```

Device-Specific Setup

Update only repositories required for a specific device family:

```
# Update only repositories for specific device, e.g., MCXW235
west update_board --set device mcxw23

# List available repositories for the device family
west update_board --set device mcxw23 --list-repo
```

Custom Configuration

For advanced users who want to create custom repository combinations:

```
# Use custom configuration file
west update_board --set custom path/to/custom-config.yml

# Generate custom configuration template
cp manifests/boards/custom.yml.template my-custom-config.yml
```

Benefits of Targeted Setup

Reduced Download Size

- Download only components needed for your target board or device
- Significantly faster initial setup for focused development
- Typical reduction from 7 GB to 2GB

Optimized Workspace

- Cleaner workspace with relevant components only
- Reduced disk space usage
- Faster repository operations

Flexible Development

- Switch between different board configurations easily
- Maintain separate workspaces for different projects
- Include optional components as needed

Repository Information

Before setting up your workspace, you can explore what repositories are available:

```
# Display repository information in console
west update_board --set board frdmmcxw23 --list-repo

# Export repository information to YAML file for reference
west update_board --set board frdmmcxw23 --list-repo -o board-repos.yml
```

This command lists all the available repositories with descriptions and outlines the included components in the workspace.

Package Generation (Optional)

The update_board command can also generate ZIP packages for offline distribution:

```
# Generate board-specific SDK package
west update_board --set board frdmmcxw23 -o frdmmcxw23-sdk.zip
```

Note: Package generation is primarily intended for creating custom SDK distributions. For regular development, use the workspace update commands without the -o option.

Workspace Management

Updating Your Workspace

Keep your SDK current with latest updates from GitHub:

For Complete SDK Workspace:

```
# Update manifest repository
cd manifests
git pull

# Update all component repositories
cd ..
west update
```

For Targeted Workspace:

```
# Update manifest repository
cd manifests
git pull

# Update board-specific repositories
cd ..
west update_board --set board your_board_name
```

Workspace Status

Check workspace synchronization status:

```
# Show status of all repositories
west status

# Show detailed information about repositories
west list
```

Troubleshooting

Network Issues:

- Use west update --keep-descendants for partial failures
- Configure Git credentials for private repositories
- Check firewall settings for Git protocol access

Permission Issues:

- Ensure write permissions in workspace directory
- Run commands without sudo/administrator privileges

- Verify Git SSH key configuration for authenticated access

Disk Space:

- Full SDK workspace requires approximately 7-8 GB
- Targeted workspace typically requires 1-2 GB
- Use board-specific setup to reduce workspace size

Repository Management Issues:

- Verify board/device names match available configurations
- Check that custom YAML files follow the correct template format
- Use `--list-repo` to verify available repositories before setup

Next Steps

With your workspace initialized:

1. Review *Workspace Structure* to understand the layout
2. Build your first project with *First Build Guide*
3. Explore *Development Workflows MCUXPresso VSCode* or *Development Workflows Command Line* for the details on project setup and execution

For advanced repository management, see the west tool documentation.

Prebuilt Binary Blobs Some SDK components ship with prebuilt binary files (`.a`, `.lib`, `.bin`, etc.) that are **not included in the git repository itself**. Instead they are fetched on demand from a dedicated blob repository using the `west blobs` command.

This keeps the main SDK checkout small and fast while still making the binaries available immediately when you need them.

When do you need to fetch blobs?

You typically need blobs when:

- You are building an example that links against a prebuilt library (e.g. a machine learning inference engine, a wireless stack library, or a graphics middleware).
- After a fresh `west init` + `west update`, the build fails with a linker error about a missing `.a` or `.lib` file.
- You run `west blobs check` and see `MISSING` entries.

Quick start

Run these commands from your SDK workspace root (the directory that contains `manifests/` and `mcuxsdk/`):

```
# 1. See what blobs are available and whether they are already present
west blobs check

# 2. Fetch any missing blobs
west blobs fetch

# 3. Confirm everything is present
west blobs check
```

A green result looks like:

```
mcu-sdk-examples: 1078/1078 entries: OK
```

Fetch blobs for a specific project only

If you only need blobs for one component, pass the west project name:

```
west blobs fetch mcu-sdk-examples
```

Use `west blobs list` to see available project names and their blob files:

```
west blobs list
```

License

By fetching blobs you accept the license terms bundled with each blob set. The applicable license file is located alongside `mcux_blobs.yml` in the relevant SDK subdirectory (for example `mcuxsdk/examples/LICENSE`).

Some blob sets require explicit license acceptance. If a blob has `click-through: True`, `west blobs fetch` will prompt you to confirm before downloading.

Keeping blobs up to date

After running `west update` (which updates the SDK source), re-run:

```
west blobs fetch
```

Blobs that are already present and whose sha256 matches are skipped automatically; only new or changed blobs are downloaded.

Troubleshooting

Build error: cannot find `-lsomething` or missing `.a` / `.bin`

```
west blobs fetch      # fetch all missing blobs
west blobs check      # verify
```

Re-fetch a corrupted or incomplete blob

```
# Remove locally cached blobs for one project and re-download
west blobs clean mcu-sdk-examples
west blobs fetch mcu-sdk-examples
```

Network / proxy issues

`west blobs fetch` downloads files over HTTPS. If your network requires a proxy, set the standard environment variables before running the command:

```
export HTTPS_PROXY=http://your-proxy:port
west blobs fetch
```

Checking without downloading

```
west blobs check      # shows OK or MISSING per entry; does not download
west blobs list       # lists all declared blobs; does not download
```

Next steps

- *Building Your First Project* — build and run an SDK example
- *Exploring the SDK* — workspace layout and example organization

Explore SDK Structure and Content

Learn about the organization of the SDK and its components:

SDK Architecture Overview The MCUXpresso SDK uses a modular architecture where software components are distributed across multiple repositories hosted on GitHub and managed through the west tool. This approach provides flexibility, maintainability, and enables selective component inclusion.

Repository Organization

Based on the manifest structure, the SDK consists of four main repository categories:

Manifest Repository

The manifest repo (mcuxsdk-manifests) contains the west.yml manifest file that tracks all other repositories in the SDK.

Base Repositories

Recorded in submanifests/base.yml and loaded in the root west.yml manifest file. These are the foundational repositories that build the SDK:

- **Devices**: MCU-specific support packages
- **Examples**: Demonstration applications and code samples
- **Boards**: Board support packages

Middleware Repositories

Recorded in the submanifests/middleware subdirectory, categorized according to functionality:

- **Connectivity**: Networking stacks, USB, and communication protocols
- **Security**: Cryptographic libraries and secure boot components
- **Wireless**: Bluetooth, IEEE 802.15.4, and other wireless protocols
- **Graphics**: Display drivers and UI frameworks
- **Audio**: Audio processing and voice recognition libraries
- **Machine Learning**: AI inference engines and neural network libraries

- **Safety**: IEC60730B safety libraries
- **Motor Control**: Motor control and real-time control libraries

Internal Repositories

Recorded in submanifests/internal.yml and grouped into the "bifrost" group. These are only visible to NXP internal developers and hosted on NXP internal git servers.

Repository Hosting

Public repositories are hosted on GitHub under these organizations:

- nxp-mcuxpresso
- NXP
- nxp-zephyr

Internal repositories are hosted on NXP's private Git infrastructure.

Benefits of This Architecture

Selective Integration: Projects include only required components, reducing memory footprint and build complexity.

Independent Versioning: Each component maintains its own release cycle and version control.

Community Collaboration: Public repositories accept community contributions through standard Git workflows.

Scalable Maintenance: Component owners can update their repositories without affecting the entire SDK.

Workspace Management

The west tool manages repository synchronization, version tracking, and workspace updates. All repositories are checked out under the mcuxsdk/ directory with their designated paths defined in the manifest files.

Workspace Structure After you initialize your SDK workspace, it creates a specific directory structure that organizes all SDK components. This structure is identical for both GitHub Repository SDK and Repository-Layout SDK Package.

Top-Level Organization

```
your-sdk-workspace/
  manifests/        # West manifest repository
  mcuxsdk/          # Main SDK content
```

The mcuxsdk/ directory serves as your primary working directory and contains all the SDK components.

SDK Component Layout

Based on the actual SDK structure, the main directories include:

Directory	Contents	Purpose
arch/	Architecture-specific files	ARM CMSIS, build configurations
cmake	Build system modules	CMake configuration and build rules
compo	Software components	Reusable software libraries and utilities
devices	Device support packages	MCU-specific headers, startup code, linker scripts
drivers	Peripheral drivers	Hardware abstraction layer for MCU peripherals
examp	Sample applications	Demonstration code and reference implementations
middle	Optional software stacks	Networking, graphics, security, and other libraries
rtos/	Operating system support	FreeRTOS integration
scripts	Build and utility scripts	West extensions and development tools
svd	Svd files for devices, this is optional because of large size. Customers run west manifest config group.filter +optional and west update mcux-soc-svd to get this folder.	

Example Organization

Examples follow a two-tier structure separating common code from board-specific implementations:

Common Example Files

```
examples/demo_apps/hello_world/
  CMakeLists.txt       # Build configuration
  example.yml          # Example metadata
  hello_world.c        # Application source code
  Kconfig              # Configuration options
  readme.md            # General documentation
```

Board-Specific Files

```
examples/_boards/your_board/demo_apps/hello_world/
  app.h                    # Board specific application header
  example_board_readme.md  # Board specific documentation
  hardware_init.c          # Board specific hardware initialization
  pin_mux.c                # Pin multiplexing configuration
  pin_mux.h                # Pin multiplexing header definitions
  hello_world.bin          # Pre-built binary for quick testing
  hello_world.mex          # MCUXpresso Config Tools project file
  prj.conf                 # Board specific Kconfig configuration
  reconfig.cmake           # Board specific cmake configuration overrides
```

Device Support Structure

Device support is organized hierarchically by MCU family:

```
devices/
  MCX/                  # MCU portfolio
    MCXW/               # MCU family
      MCXW235/          # Specific device
        MCXW235.h           # Device register definitions
        drivers/            # Device-specific drivers
        gcc/                # GNU toolchain files
        iar/                # IAR toolchain files
        mcuxpresso/         # MCUXpresso IDE files
        startup_MCXW235.c # Startup and vector table
        system_MCXW235.c  # System initialization
```

Middleware Organization

Middleware components are categorized by functionality and maintained in separate repositories. Based on the manifest files, common middleware categories include:

- **Connectivity**: USB, TCP/IP, industrial protocols
- **Security**: Cryptographic libraries, secure boot
- **Wireless**: Bluetooth, IEEE 802.15.4, Wi-Fi
- **Graphics**: Display drivers, UI frameworks
- **Audio**: Processing libraries, voice recognition
- **Machine Learning**: Inference engines, neural networks
- **Safety**: IEC60730B safety libraries
- **Motor Control**: Motor control and real-time control libraries

Documentation Structure

SDK documentation is distributed across multiple locations:

- docs/ - Core SDK documentation and build infrastructure
- Component repositories - API documentation and integration guides
- Board directories - Hardware-specific setup instructions

For complete documentation, refer to the online documentation.

Understanding Example Structure

Each example has **two README files**:

1. General README: examples/demo_apps/hello_world/readme.md

- What the example does
- General functionality description
- Common usage information

2. Board-Specific README: examples/_boards/{board_name}/demo_apps/hello_world/example_board_readme.md

- Board-specific setup instructions
- Hardware connections required

- Board-specific behavior notes

Tip: Always check both readme files - start with the general one, then read the board-specific one for detailed setup.

Development Workflows

Get started with building and running projects:

Building Your First Project This guide explains how to build and run your first SDK example project using the west build system. This applies to both GitHub Repository SDK and Repository-Layout SDK Package.

Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- Development board connected via USB
- Build tools installed per *Installation Guide*

Understanding Board Support

Before building, check whether your board is single-core or multicore, and which core IDs it supports:

```
west boards -b your_board
```

This prints the resolved device and, for multicore boards, the list of valid `core_ids` you'll need for the build command below. For a single-core board, no core ID is needed.

If you want to see every supported build configuration (toolchain/config combination) for a specific example instead, use:

```
west list_project -p examples/demo_apps/hello_world
```

This shows all supported build configurations. You can filter by toolchain:

```
west list_project -p examples/demo_apps/hello_world -t armgcc
```

`west list_project` is the most complete way to discover build commands across all boards and targets, but `west boards -b your_board` is the quicker way to look up just the board/core information you need once you already know which example you want to build.

Basic Build Process

Simple Build

Build the hello_world example with default settings:

```
west build -b your_board examples/demo_apps/hello_world
```

The default toolchain is armgcc, and the build system will select the first debug target as default if no config is specified.

Specifying Configuration

```
# Release build
west build -b your_board examples/demo_apps/hello_world --config release

# Debug build (default)
west build -b your_board examples/demo_apps/hello_world --config debug
```

Alternative Toolchains

```
# IAR toolchain
west build -b your_board examples/demo_apps/hello_world --toolchain iar

# Other toolchains as supported by the example
```

Multicore Applications

For multicore devices, west boards -b your_board (above) tells you which core IDs are valid. Specify the core either with -Dcore_id=:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config␣
↪flexspi_nor_debug
```

or more concisely by appending @<core_id> to the board name:

```
west build -b evkbmimxrt1170@cm7 examples/demo_apps/hello_world --toolchain iar --config flexspi_nor_
↪debug
```

Single-core boards don't need a core ID at all.

For multicore projects using sysbuild:

```
west build -b evkbmimxrt1170 --sysbuild ./examples/multicore_examples/hello_world/primary -Dcore_
↪id=cm7 --config flexspi_nor_debug --toolchain=armgcc -p always
```

Flash an Application

Flash the built application to your board:

```
west flash -r linkserver
```

Debug

Start a debug session:

```
west debug -r linkserver
```

Common Build Options

Clean Build

Force a complete rebuild:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Dry Run

See the commands that get executed without running them:

```
west build -b your_board examples/demo_apps/hello_world --dry-run
```

Device Variants

For boards supporting multiple device variants:

```
west build -b your_board examples/demo_apps/hello_world --device DEVICE_PART_NUMBER --config␣
↪release
```

Project Configuration

CMake Configuration Only

Run configuration without building:

```
west build -b your_board examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

Interactive Configuration

Launch the configuration GUI:

```
west build -t guiconfig
```

Troubleshooting

Build Failures

Use pristine builds to resolve dependency issues:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Getting Help

View the help information for west build:

```
west build -h
```

Check Supported Configurations

To see available configuration options and board targets for an example, refer to the below command:

```
west list_project -p examples/demo_apps/hello_world
```

Next Steps

- Explore other examples in the SDK
- Learn about *Command Line Development* for advanced options
- Try *VS Code Development* for integrated development
- Refer *Workspace Structure* to understand the SDK layout

MCUXpresso for VS Code Development This guide covers using MCUXpresso for VS Code extension to build, debug, and develop SDK applications with an integrated development environment.

Prerequisites

- SDK workspace initialized (GitHub Repository SDK or Repository-Layout SDK Package)
- Development tools installed per *Installation Guide*
- Visual Studio Code installed
- MCUXpresso for VS Code extension installed

Extension Installation

Install MCUXpresso for VS Code

The MCUXpresso for VS Code extension provides integrated development capabilities for MCUXpresso SDK projects. Refer to the MCUXpresso for VS Code documentation for detailed installation and setup instructions.

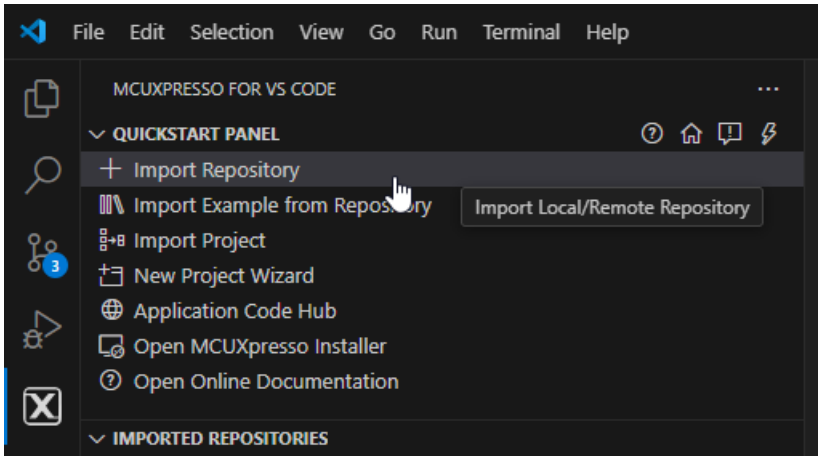
SDK Import and Setup

Import Methods

The SDK can be imported in several ways. The MCUXpresso for VS Code extension supports both GitHub Repository SDK and Repository-Layout SDK Package distributions.

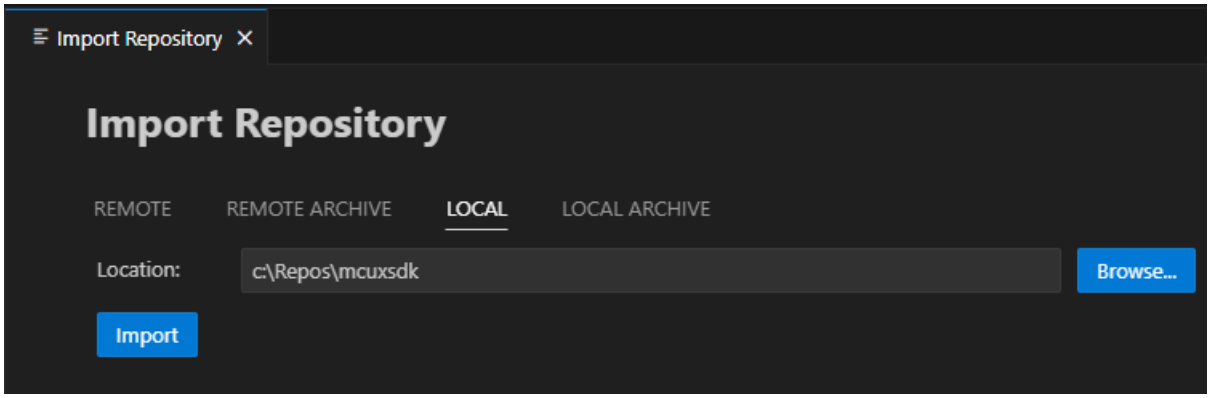
Import GitHub Repository SDK

Click **Import Repository** from the **QUICKSTART PANEL**

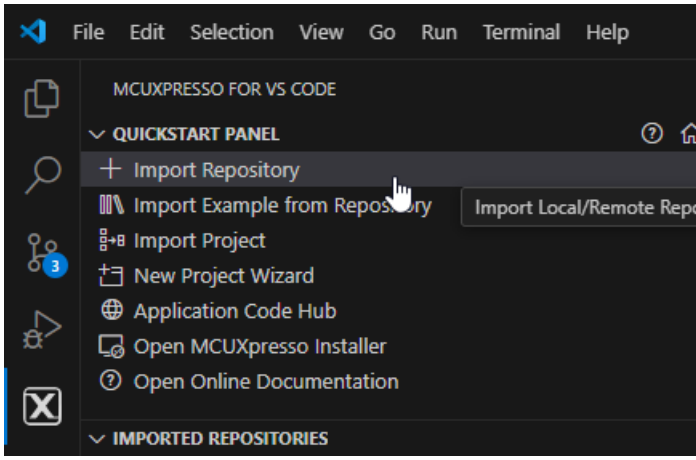


Note: You can import the SDK in several ways. Refer to the MCUXpresso for VS Code documentation for details.

Select **Local** if you've already obtained the SDK according to *setting up the repo*. Select your location and click **Import**.

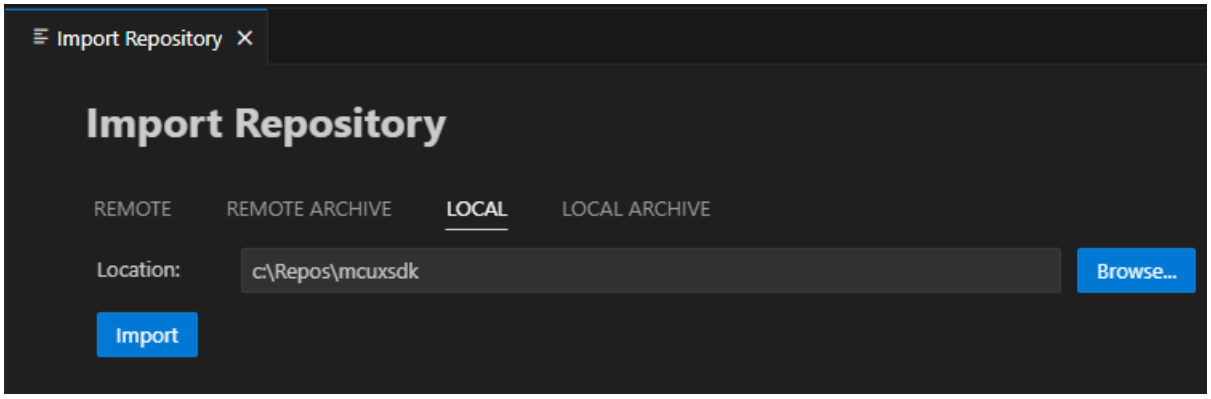


Import Repository-Layout SDK Package

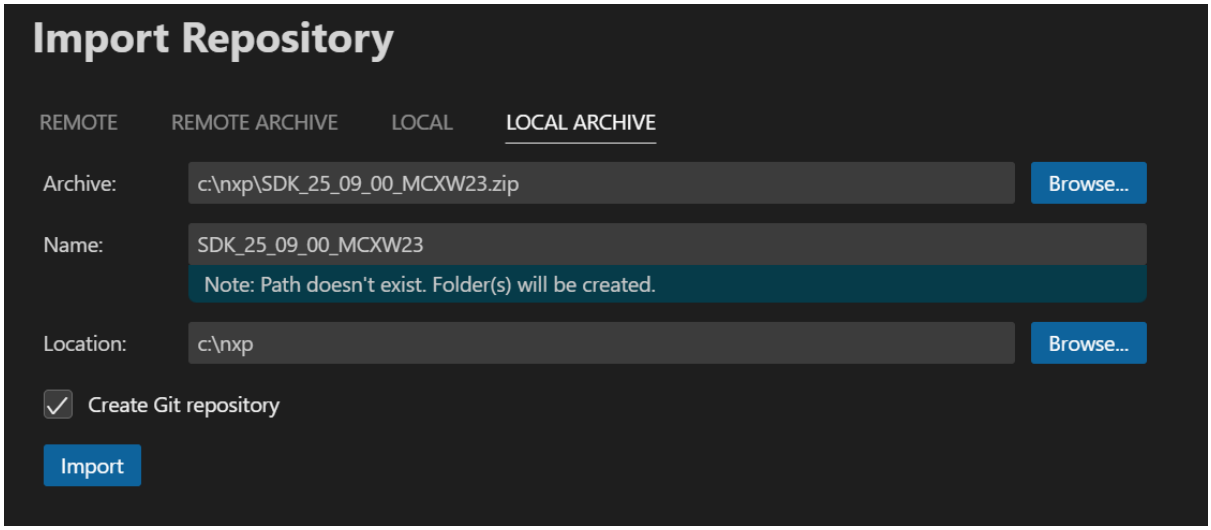


Click **Import Repository** from the **QUICKSTART PANEL**

Select **Local** if you've already unzipped the Repository-Layout SDK Package. Select your location and click **Import**.



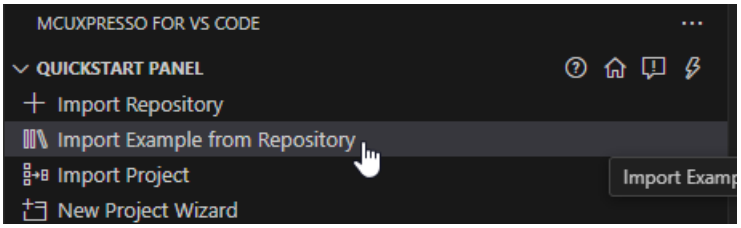
Else if the SDK is ZIP archive, select **Local Archive**, browse to the downloaded SDK ZIP file, fill the link of expect location, then click **Import**.



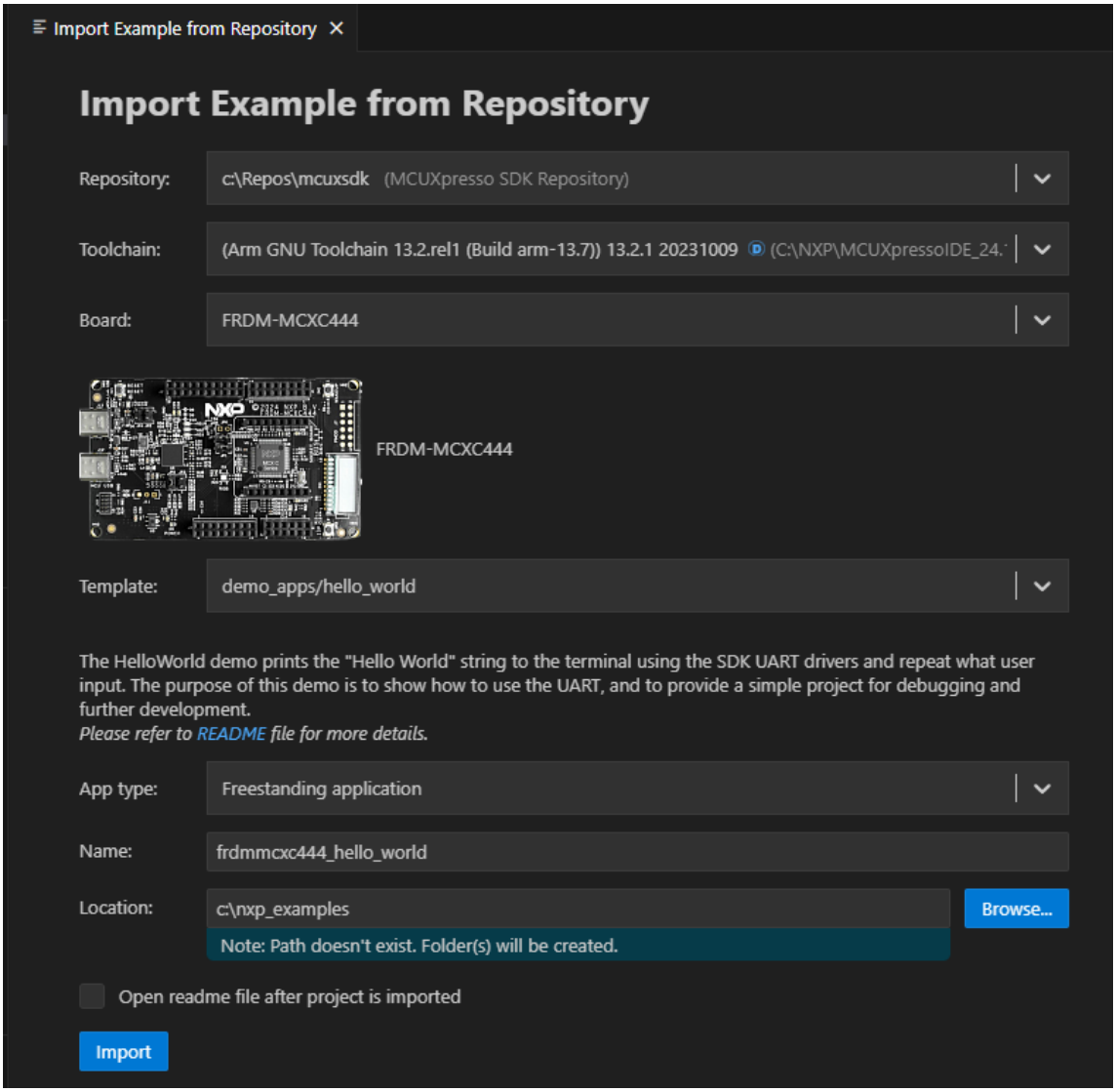
Building Example Applications

Import Example Project

1. Click **Import Example from Repository** from the **QUICKSTART PANEL**



2. Configure project settings:
   - **MCUXpresso SDK**: Select your imported SDK
   - **Arm GNU Toolchain**: Choose toolchain
   - **Board**: Select your target development board
   - **Template**: Choose example category
   - **Application**: Select specific example (e.g., hello_world)
   - **App type**: Choose between Repository applications or Freestanding applications
3. Click **Import**



Application Types

Repository Applications:

- Located inside the MCUXpresso SDK
- Integrated with SDK workspace

Freestanding Applications:

- Imported to user-defined location
- Independent of SDK location

Trust Confirmation

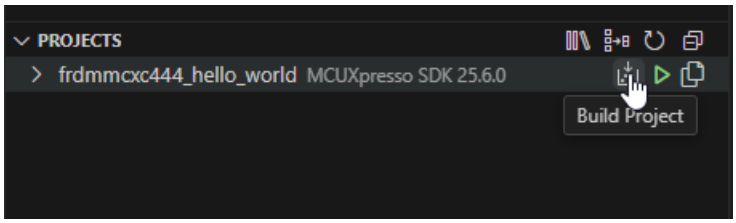
VS Code will prompt you to confirm if the imported files are trusted. Click **Yes** to proceed.

Building Projects

Build Process

1. Navigate to **PROJECTS** view
2. Find your project

3. Click the **Build Project** icon

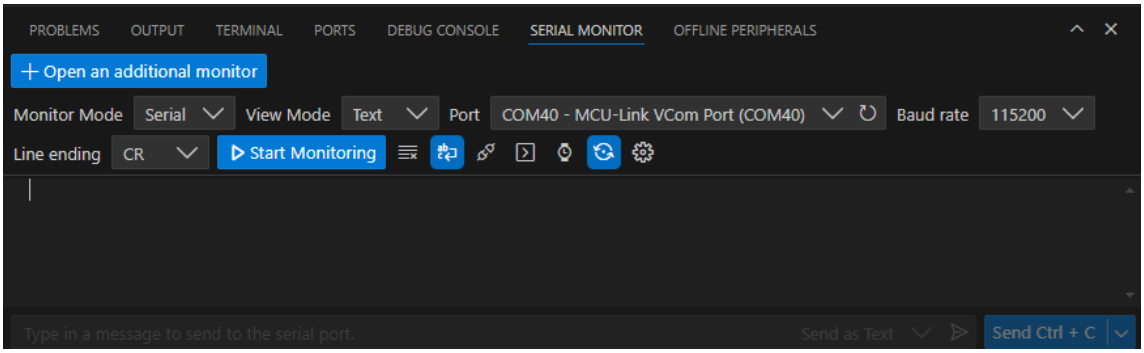


The integrated terminal will display build output at the bottom of the VS Code window.

Running and Debugging

Serial Monitor Setup

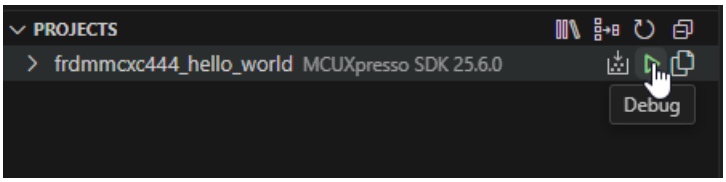
1. Open **Serial Monitor** from VS Code's integrated terminal



2. Configure serial settings:
   - **VCom Port**: Select port for your device
   - **Baud Rate**: Set to 115200

Debug Session

1. Navigate to **PROJECTS** view
2. Click the play button to initiate a debug session



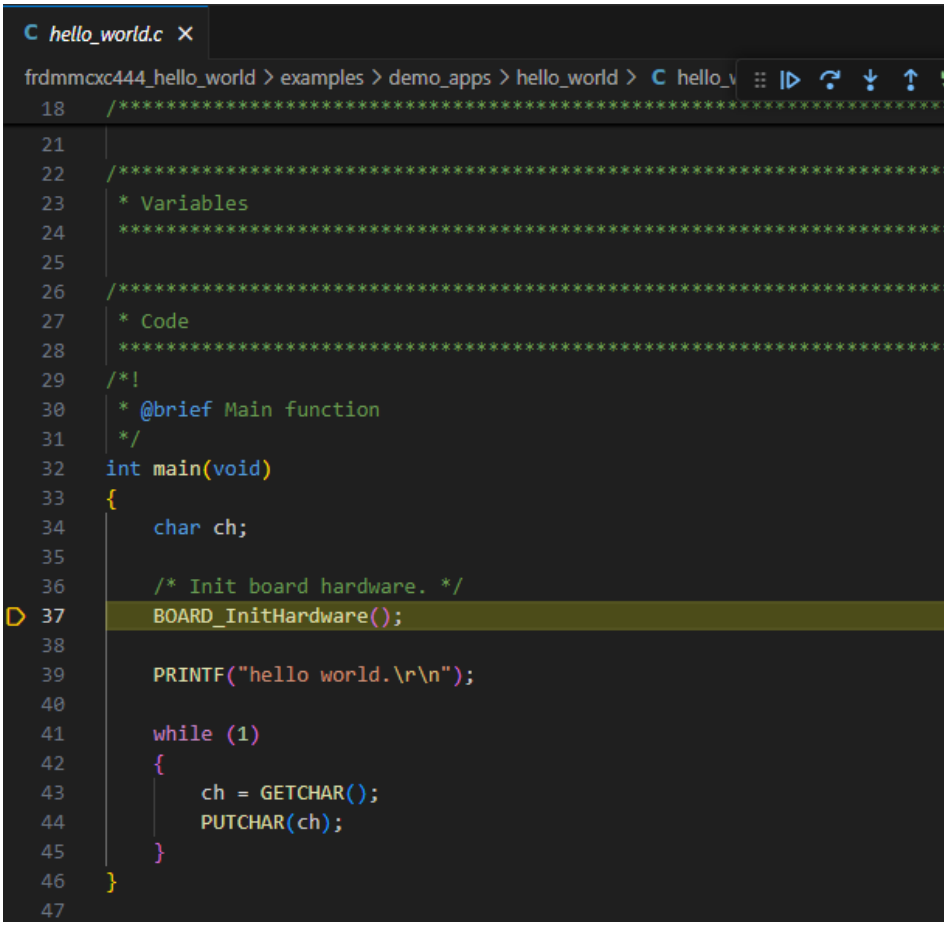
The debug session will begin with debug controls initially at the top of the interface.

Debug Controls

Use the debug controls to manage execution:

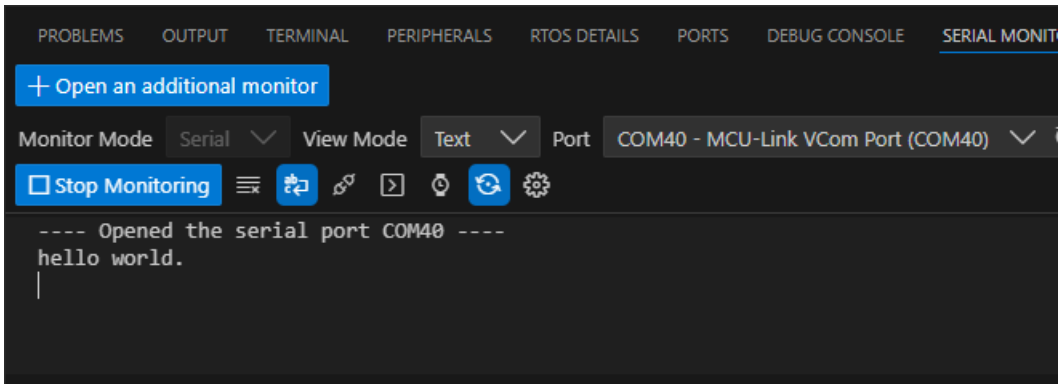
- **Continue**: Resume code execution
- **Step controls**: Navigate through code

- **Stop**: Terminate debug session



Monitor Output

Observe application output in the **Serial Monitor** to verify correct operation.



Debug Probe Support

For comprehensive information on debug probe support and configuration, refer to the MCUXpresso for VS Code Debug documentation.

Project Configuration

Workspace Management

The extension integrates with the MCUXpresso SDK workspace structure, providing access to:

- Example applications

- Board configurations
- Middleware components
- Build system integration

Multi-Project Support

The PROJECTS view allows management of multiple imported projects within the same workspace.

Troubleshooting

Import Issues

SDK not detected:

- Verify SDK workspace is properly initialized
- Ensure all required repositories are updated
- Check SDK manifest files are present

Project import failures:

- Confirm board support exists for selected example
- Verify toolchain installation
- Check example compatibility with selected board

Build Problems

Build failures:

- Check integrated terminal for error messages
- Verify all dependencies are installed
- Ensure toolchain is properly configured

Debug Issues

Debug session fails:

- Verify board connection via USB
- Check debug probe drivers are installed
- Confirm build completed successfully

Serial monitor problems:

- Verify correct VCom port selection
- Check baud rate configuration (115200)
- Ensure board drivers are installed

Integration with Command Line

MCUXpresso for VS Code integrates with the underlying west build system, allowing seamless integration with command line workflows described in *Command Line Development*.

Advanced Features

Project Types

The extension supports both repository-based and freestanding project types, providing flexibility in project organization and SDK integration.

Build System Integration

The extension leverages the MCUXpresso SDK build system, providing access to all build configurations and options available through command line tools.

Next Steps

- Explore additional examples in the SDK
- Review *Command Line Development* for advanced build options
- Refer to the MCUXpresso for VS Code documentation for detailed documentation
- Learn about *SDK Architecture* for better understanding of the development environment

Command Line Development This guide covers developing with the MCUXpresso SDK using command line tools and the west build system. This workflow applies to both GitHub Repository SDK and Repository-Layout SDK Package distributions.

Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- Development tools installed per *Installation Guide*
- Target board connected via USB

Understanding Board Support

Use the west extension to discover available examples for your board:

```
west list_project -p examples/demo_apps/hello_world
```

This shows all supported build configurations. You can filter by toolchain:

```
west list_project -p examples/demo_apps/hello_world -t armgcc
```

Basic Build Commands

Standard Build Process

Build with default settings (armgcc toolchain, first debug config):

```
west build -b your_board examples/demo_apps/hello_world
```

Specifying Build Configuration

```
# Release build
west build -b your_board examples/demo_apps/hello_world --config release

# Debug build with specific toolchain
west build -b your_board examples/demo_apps/hello_world --toolchain iar --config debug
```

Multicore Applications

For multicore devices, specify the core ID:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config␣
↪flexspi_nor_debug
```

For multicore projects using sysbuild:

```
west build -b evkbmimxrt1170 --sysbuild ./examples/multicore_examples/hello_world/primary -Dcore_
↪id=cm7 --config flexspi_nor_debug --toolchain=armgcc -p always
```

Shield Support

For boards with shields:

```
west build -b mimxrt700evk --shield a8974 examples/issdk_examples/sensors/fxls8974cf/fxls8974cf_poll -
↪Dcore_id=cm33_core0
```

Advanced Build Options

Clean Builds

Force a complete rebuild:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Dry Run

See what commands would be executed:

```
west build -b your_board examples/demo_apps/hello_world --dry-run
```

Device Variants

For boards supporting multiple device variants:

```
west build -b your_board examples/demo_apps/hello_world --device MK22F12810 --config release
```

Project Configuration

CMake Configuration Only

Run configuration without building:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world -Dcore_id=cm7 --cmake-only -p
```

Interactive Configuration

Launch the configuration GUI:

```
west build -t guiconfig
```

Flashing and Debugging

Flash Application

Flash the built application to your board:

```
west flash -r linkserver
```

Debug Session

Start a debugging session:

```
west debug -r linkserver
```

IDE Project Generation

To use a traditional IDE instead of the command line or VS Code, generate an IDE project from the CMake build with `-t guiproject`:

- Run a demo using IAR
- Run a demo using MDK

Both guides cover the Ruby dependency required for IDE project generation.

Troubleshooting

Build Failures

Use pristine builds to resolve dependency issues:

```
west build -b your_board examples/demo_apps/hello_world -p always
```

Toolchain Issues

Verify environment variables are set correctly:

```
# Check ARM GCC
echo $ARMGCC_DIR
arm-none-eabi-gcc --version

# Check IAR (if using)
echo $IAR_DIR
```

Getting Help

Display help information:

```
west build -h
west flash -h
west debug -h
```

Check Supported Configurations

If unsure about supported options for an example:

```
west list_project -p examples/demo_apps/hello_world
```

Best Practices

Project Organization

- Keep custom projects outside the SDK tree
- Use version control for your application code
- Document any SDK modifications

Build Efficiency

- Use `-p always` for clean builds when troubleshooting
- Leverage `--dry-run` to understand build processes
- Use specific configs and toolchains to reduce build time

Development Workflow

1. Start with existing examples closest to your requirements
2. Copy and modify rather than building from scratch
3. Test with hello_world before moving to complex examples
4. Use configuration tools for pin muxing and clock setup

Next Steps

- Explore *VS Code Development* for integrated development experience
- Review *Workspace Structure* to understand SDK organization
- Refer build system documentation for advanced configurations

Using MCUXpresso Config Tools MCUXpresso Config tools provide a user-friendly way to configure hardware initialization of your projects. This guide explains the basic workflow with the MCUXpresso SDK west build system and the Config Tools.

Prerequisites

- GitHub Repository SDK workspace initialized OR Repository-Layout SDK Package extracted
- MCUXpresso Config Tools standalone installed (version 25.09 or above)
- MCUXpresso SDK Project that can be successfully built

Board Files

MCUXpresso Config Tools generate source files for the board. These files include pin_mux.c/h and clock_config.c/h. The files contain initialization code functions that reflect the hardware configuration in the Config Tools. Within the SDK codebase, these files are specific for the board and either shared by multiple example projects or specific for one example. Open or import the configuration from the SDK project in the Config Tools and customize the settings to match the custom board or specific project use case and regenerate the code. See *User Guide for MCUXpresso Config Tools (Desktop)* (document GSMCUXCTUG) for details.

Note: When opening the configuration for SDK example projects, the board files may be shared across multiple examples. To ensure a separate copy of the board configuration files exists, create a freestanding project with copied board files.

Visual Studio Code

To open the configuration in Visual Studio Code, use the context menu for the project to access Config Tools. See MCUXpresso Extension Documentation for details.
Otherwise, use the manual workflow described in detail in the following section.

Manual Workflow

Use the following steps:

1. Before using Config Tools, run the west command to get the project information for Config Tools from the SDK project files, for example:

```
west cfg_project_info -b lpcxpresso55s69 ...mcuxsdk/examples/demo_apps/hello_world/ -Dcore_
↪id=cm33_core0
```

This results in the creation of the project information json file that is searched by the config tools when the configuration is created. The parameters of the command should match the build parameters that will be used for the project.

2. Launch the MCUXpresso Config Tools and in the **Start development** wizard, select **Create a new configuration based on the existing IDE/Toolchain project**. Select the created "cfg_tools" subfolder as a project folder (for example: ...mcuxsdk/examples/demo_apps/hello_world/cfg_tools/).

Updating the SDK West project

Note: Updating project is supported with Config Tools V25.12 or newer only.

Changes in the Config tools generated source code modules may require adjustments to the toolchain project to ensure a successful build. These changes may mean, for example, adding the newly generated files, adding include paths, required drivers, or other SDK components. This section describes how to manually resolve the changes needed in the project within the toolchain projects based on the SDK project managed by the West tool.

After the configuration in the Config Tools is finished, write updated files to the disk using the 'Update Code' command. The written files include a json file with the required changes for the toolchain project.

To resolve the changes in the project in the terminal, launch the west command that updates the project. For example:

```
west cfg_resolve -b lpcxpresso55s69 ...mcuxsdk/examples/demo_apps/hello_world/ -Dcore_id=cm33_core0
```

This command updates the appropriate cmake and kconfig files to address the changes. After this, the application can be built.

Note: The cfg_resolve command supports additional arguments. Launch the *west cfg_resolve -h* command to get the list and description.

1.5 Getting Started with MCUXpresso SDK Xplorer

1.5.1 Getting Started with MCUXpresso SDK Xplorer

Install Xplorer Toolchains

This section provides information on Xtensa Software Tools Platform Support and steps to:

- Install the Xtensa Xplorer IDE and Tools
- Install License Key
- Install RT700 DSP Build Configuration
- Install Xtensa On Chip Debugger Daemon
- Program MCU-Link Debug Probe with SEGGER J-Link (Optional)
- Install Xtensa Software Tools without IDE

Xtensa Software Tools Platform Support The Xtensa Software Tools are officially supported on the following platforms:

Windows: Win 10 64-bit, Win 8 64-bit, Win 7 64-bit.

Linux: RHEL 6 64-bit (with 'Desktop' package installed).

There may be compatibility issues with other versions of Linux or Windows, especially when using the IDE.

Note: The security-enhanced Linux (SELinux) is not a supported platform because the OS can prevent different shared libraries (including Xtensa Tools) from loading.

For more information on platform support and installation guidelines, see the Xtensa Development Tools Installation Guide.

Install the Xtensa Xplorer IDE and Tools To install the Xtends Xplorer IDE and tools, perform the following steps:

1. Go to the URL https://tensilicatools.com/download/hifi-dsp-development-tools-for-rt700/ and log in. If you are accessing the site for the first time, make sure to register.
2. Make sure to use your corporate email address to register.

3. Once registered you should receive an email confirmation with an activation link from 'Tensilica Tools' no-reply@tensilicatools.com. Make sure to check the spam folder if this email does not show up in the inbox.
4. To complete the registration, click the activation link.
5. Once registered, log in to see the material available for download:

1. Download and install the **Xplorer IDE V10.1.11** for your operating system (Windows or Linux).
2. Download the **DSP Configuration** for your operating system – installed later through the IDE, see Install RT700 DSP Build Configuration. **Note:** NXP recommends version **10.1.11** of the Xtensa Xplorer IDE and tools for use with the RT700 DSP.

Parent topic:*Install Xplorer Toolchains*

Install License Key Xtensa development tools use FLEXlm for license management. FLEXlm licensing is required for tools such as the Xtensa Xplorer IDE, Xtensa C and C++ compiler.

Currently RT700 supports node-locked license for Xtensa tools. A node-locked license permits tools to run on a specific computer, tied to the MAC address of the primary network interface permanently attached to the machine.

Identify PC MAC Address

To generate the correct license file, you should first identify the appropriate MAC for the computer you plan to run Xtensa tools on. Remove '-' or ':' symbols in the MAC address.

Windows:

Linux:

Note: For Linux: MAC address must be associated with eth0 interface. If not, FLEXlm cannot perform the license checkout, compilation, or simulation of code is not possible. If the host has the MAC address associated with another interface, for example em1, the following approach may be used.

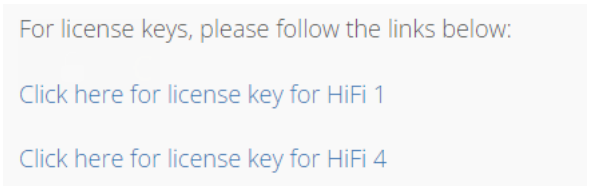
```
# Add udev rule for naming interface
$ sudo vim /etc/udev/rules.d/70-persistent-net.rules
# udev rule (replace 'XX' with the MAC address of your PC):
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="XX:XX:XX:XX:XX:XX", NAME="eth0"
# Change "em1" to "eth0" in your interfaces file.
$ sudo vim /etc/network/interfaces
# Restart udev or reboot machine
$ sudo reboot
```

Alternatively, you can use the approach recommended by your IT team to rename the interface to eth0.

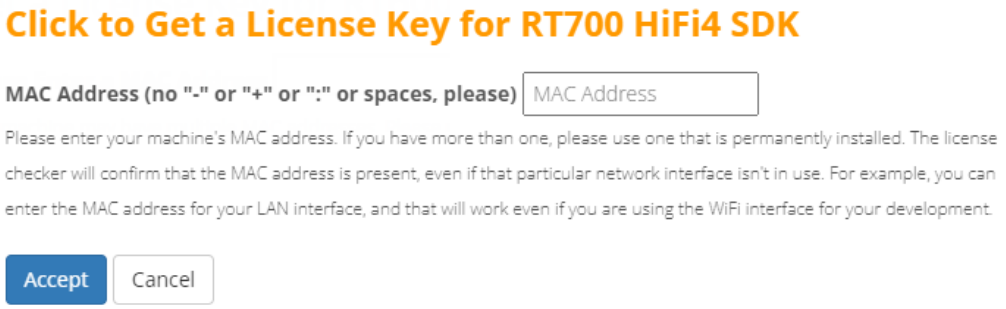
Download License Key

To download the license key:

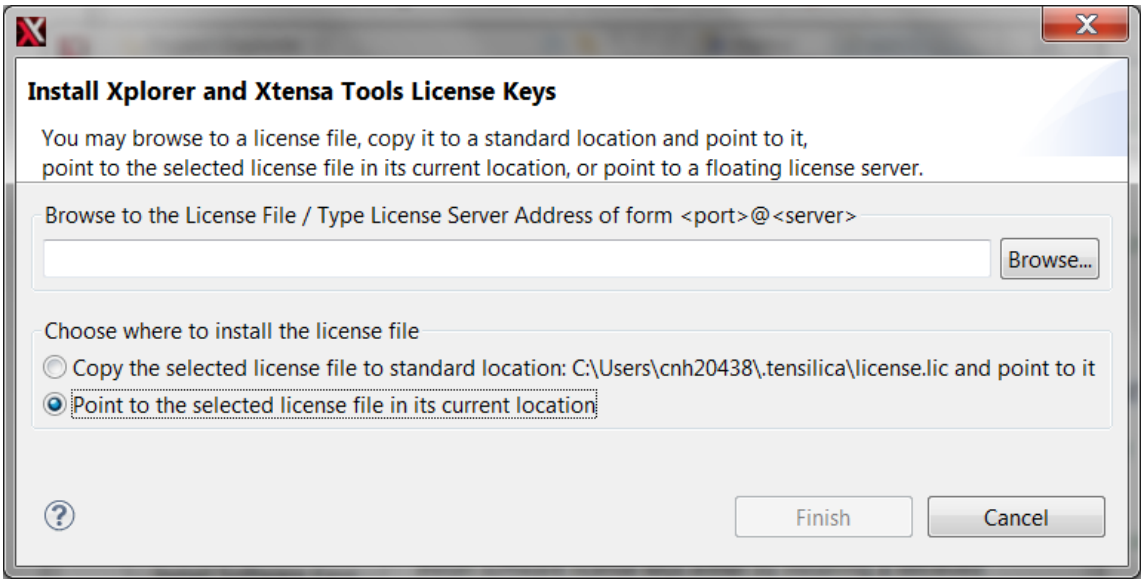
1. Reload or return to the Tensilica URL: https://tensilicatools.com/download/hifi-dsp-development-tools-for-rt700/.
2. Click the button and get the license key for RT700 SDK.



3. Provide the address.



4. Once the license file has been generated and downloaded, open your recently installed Xplorer V10.1.11.
5. Select menu **Help** > **Xplorer License Keys** > **Install Software Keys**.
6. Select the license key file.



7. Click the **Finish** button.

   Note: The generated license file only supports debug/run on the RT700 device target. It does not support software simulation/Xplorer ISS. If you have special requirements to run the software simulations, contact Cadence directly.

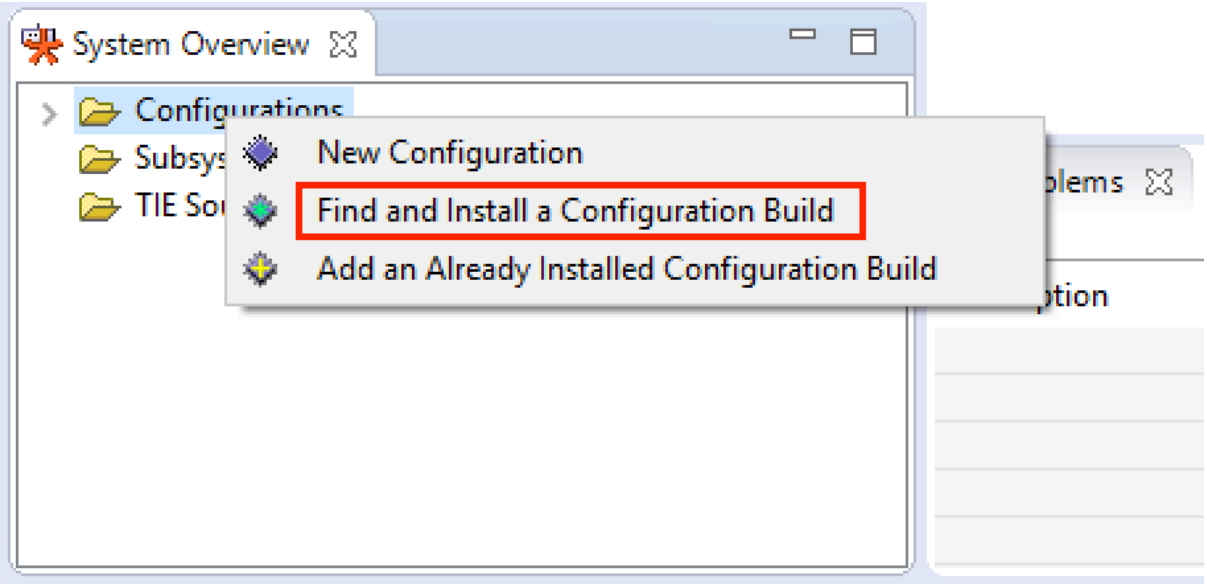
Parent topic: Install License Key

Parent topic:*Install Xplorer Toolchains*

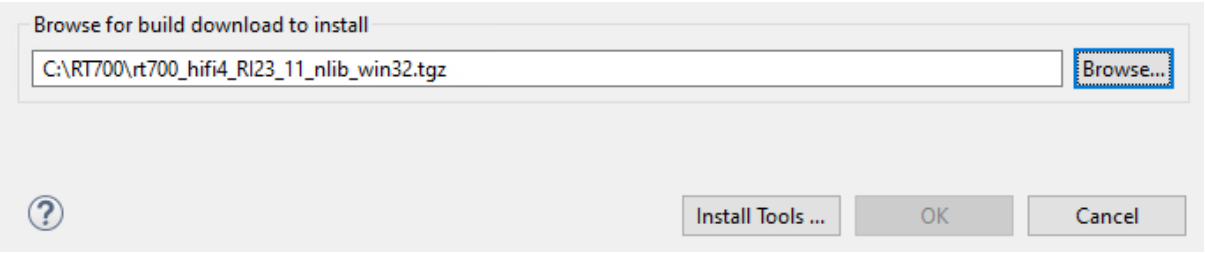
Install RT700 DSP Build Configuration *'Build Configuration'* is a term that describes all parameters and necessary build includes for the Tensilica processor implementation for development. It is mandatory to install a specific build configuration before starting development on RT700.

The build configuration is provided by NXP as a binary file and is imported into the Xplorer IDE. The binary file for the OS is available at the Tensilica URL: https://www.cadence.com/en_US/home/tools/ip/tensilica-ip/hifi-dsps/hifi-5.html.

The build configuration is installed into the IDE using the **System Overview** panel available in the lower left corner, by default. If the **System Overview** panel is not visible, toggle using the **Window** > **Show View** > **System Overview** menu item.



Click **OK** for build download to install.



Parent topic:*Install Xplorer Toolchains*

Install Xtensa On Chip Debugger Daemon The Xtensa On Chip Debugger Daemon (xt-ocd), is a gdb-based debugging tool but is not installed by default with the Xplorer IDE. For installation, a self-extracting executable installer is included with the IDE, which can be found at the following location:

Windows:

C:\usr\xtensa\XtDevTools\downloads\RI-2023.11\tools\xt-ocd-14.11-windows64-installer.exe

Linux:

~/xtensa/XtDevTools/downloads/RI-2023.11/tools/xt-ocd-14.11-linux64-installer

Currently, xt-ocd supports J-Link and Arm RVI/DSTREAM probes over Serial Wire Debug (SWD) for RT700. xt-ocd installs support for J-Link probes but does not install the required J-Link drivers which must be installed separately. Make sure that the latest version of J-Link software is installed.

Note: For Linux: When installing xt-ocd on Linux, ensure that the symlink is manually added to the installed J-Link driver: ln -s <jlink-install-dir>libjlinkarm.so.6 <xocd-install-dir>/modules/libjlinkarm.so.6.

xt-ocd is configured with an XML input file 'topology.xml' that modifies to fit the debugger hardware. Using J-link as an example, use the content below to replace the original template for HiFi4 DSP

Note: It is mandatory to replace the 'usbser' section with the J-Link serial number on the back of the J-Link hardware in use.

```
<configuration>
 <controller id='Controller0' module='jlink' usbser='600116164' type='swd' speed='1000000' locking='1'/>
 <driver id='XtensaDriver0' dap='1' xdm-offset='0x80020000' module='xtensa' step-intr='mask,stepover,setps
↪' />
 <chain controller='Controller0'>
 <tap id='TAP0' irwidth='4' />
 </chain>
 <system module='jtag'>
 <component id='Component0' tap='TAP0' config='trax' />
 </system>
 <device id='Xtensa0' component='Component0' driver='XtensaDriver0' ap-sel='3'/>
 <application id='GDBStub' module='gdbstub' port='20000' sys-reset='0'>
 <target device='Xtensa0' />
 </application>
</configuration>
```

Below is another topology.xml example for HiFi1 DSP.

```
<configuration>
      <controller id='Controller0' module='jlink' usbser='600116164' type='swd' speed='1000000' locking='1
↪'/>

  <driver id='XtensaDriver0' dap='1' xdm-offset='0x80030000' module='xtensa' step-intr='mask,stepover,
↪setps' />
  <chain controller='Controller0'>
    <tap id='TAP0' irwidth='4' />
  </chain>
  <system module='jtag'>
    <component id='Component0' tap='TAP0' config='trax' />
  </system>
  <device id='Xtensa0' component='Component0' driver='XtensaDriver0' ap-sel='3'/>
  <application id='GDBStub' module='gdbstub' port='20000' sys-reset='0'>
    <target device='Xtensa0' />
  </application>
</configuration>
```

Congratulations! All Xplorer toolchains are installed.

For more details on Xtensa software tools, build configurations, or xt-ocd daemon, see the full set of documents in Xplorer menu **Help > PDF Documentation**.

Parent topic:*Install Xplorer Toolchains*

Program MCU-Link Debug Probe with SEGGER J-Link (Optional) In addition to standalone probes, the onboard MCU-Link debug probe can be used to debug HiFi4 DSP over SWD port. The RT700 EVK has an LPC55S69 MCU (top right corner on EVK) and by default has been pre-programmed as CMSIS-DAP probe. CMSIS-DAP is not compatible with HiFi4 the DSPs, but to enable serial logging of Core 1, CMSIS-DAP firmware is required.

To use the on-board J-Link without Core 1 serial logging, a few extra steps are required to flash it with J-Link firmware.

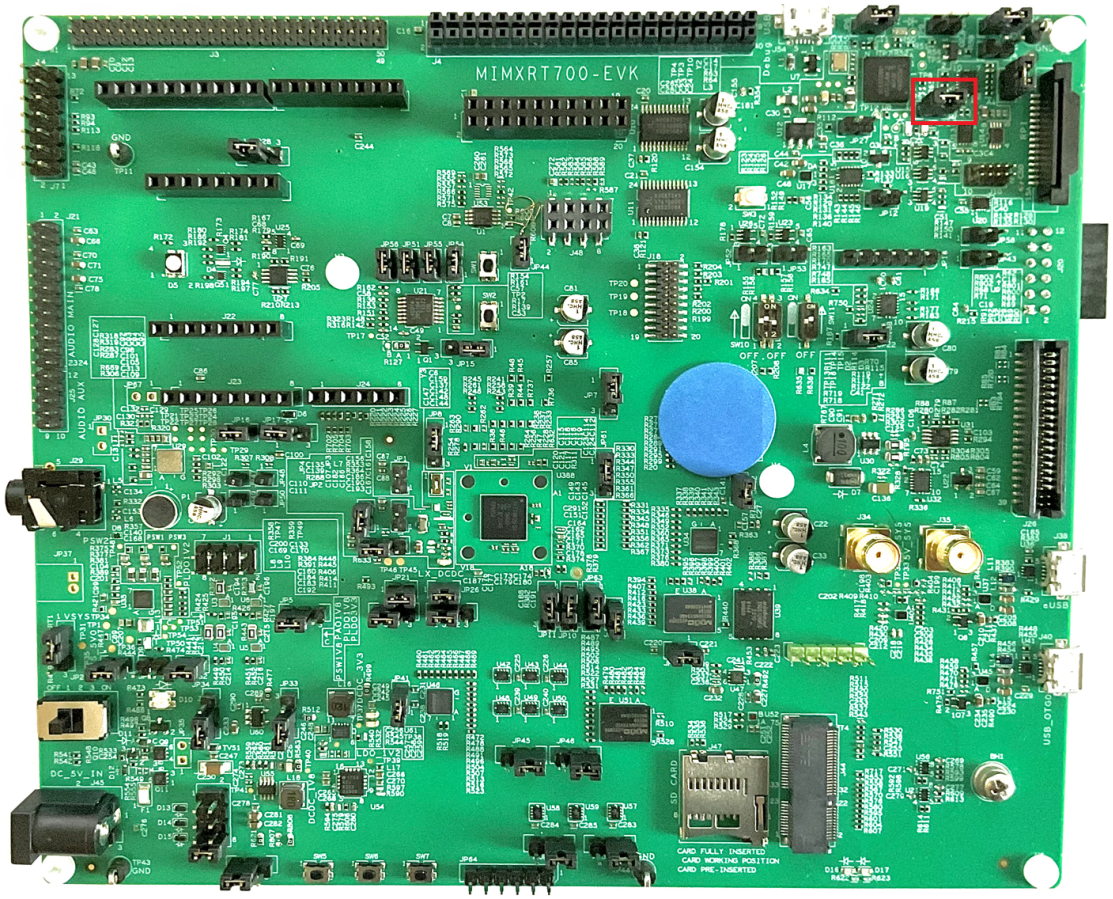
The steps are:

1. Install MCU-Link, a command-line tool for programming onboard MCU-Link debug probe with the CMSIS-DAP and J-Link firmware. MCU-Link installer is available for download from: https://www.nxp.com/design/design-center/software/development-software/mcuxpresso-software-and-tools-/mcu-link-debug-probe:MCU-LINK#t996.

2. After the download is complete, run the installer.

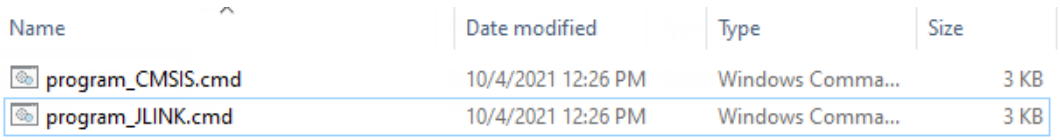
   Note: During the installation, the DFU, and VCOM drivers are automatically installed for all platforms.

3. To update the MCU-Link debug circuit firmware, unplug the USB cable on J54 and then connect the JP20 jumper. .



4. Reconnect the board to the host computer over the debug link USB connector J54.

5. Go to the scripts folder in the MCU-Link install location: C:\NXP\MCU-LINK_installer_2.263\scripts.

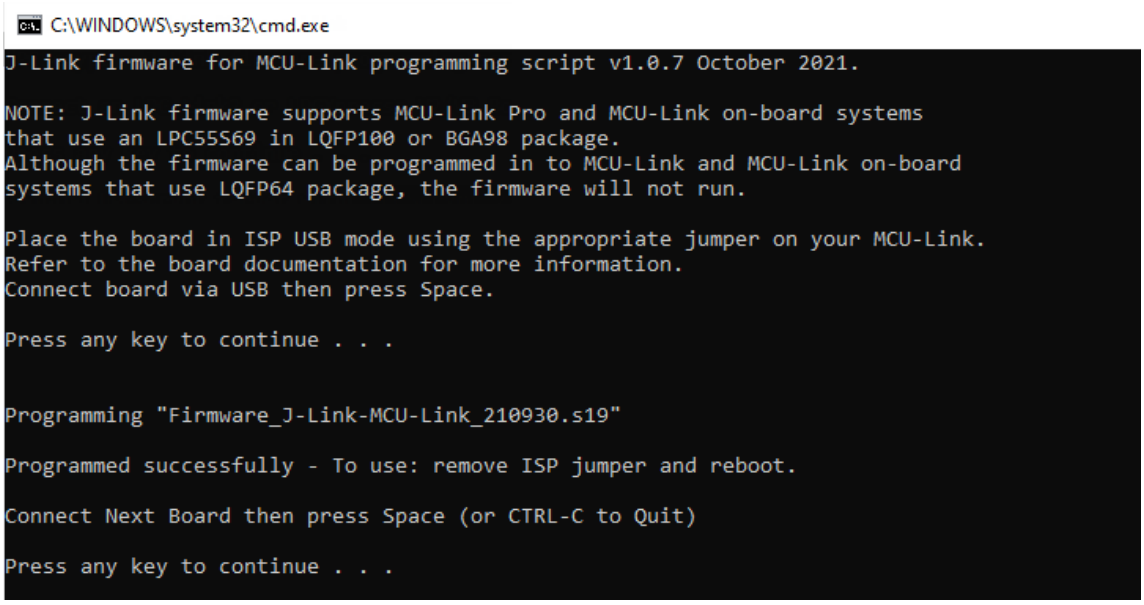


6. In that command shell, run the program_JLINK script to install the JLink debug firmware.

```
<Intall Dir>scripts\program_JLINK
```

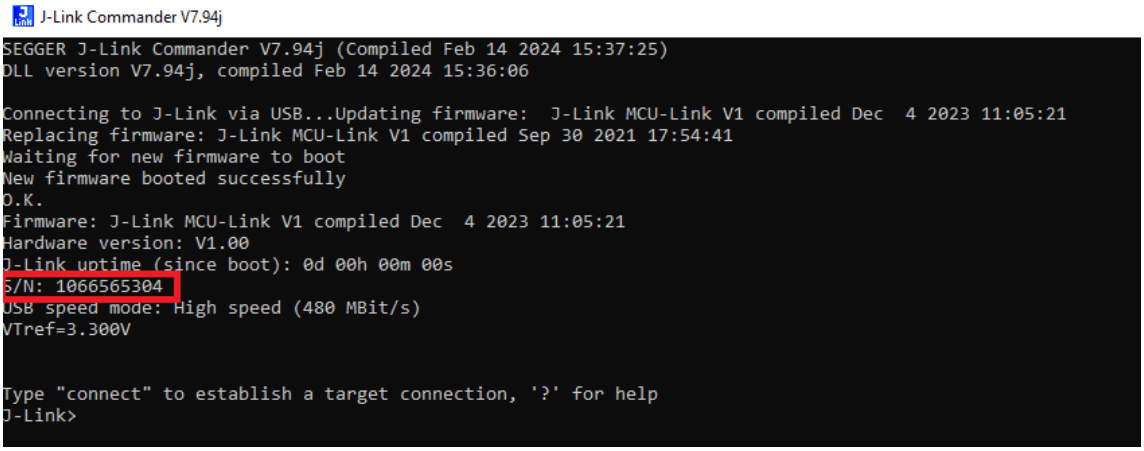
   Note: File paths in this document use the Windows directory separator, on Linux or Mac OSX. The file paths must be replaced with '/.' For Windows users, shortcuts to the scripts are available from the MCU-LInk entry on the Start menu.

7. Verify once you select the firmware (in this case J-Link), MCU-Link. The console appears as shown in Figure 4.

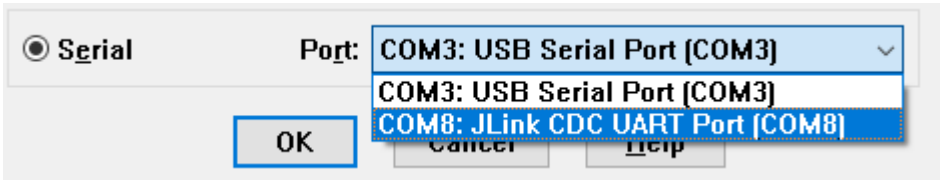


8. Open/ Disconnect JP20 and power cycle the board. The onboard MCU-Link is ready to be used as SEGGER J-Link probe.

   Every EVK/ MCU-Link has a different J-Link S/N. Therefore, make sure to write down the S/N for xt-ocd and topology.xml as indicated in Install Xtensa On Chip Debugger Daemon.



9. MCU-Link debug probe creates a virtual serial port over USB, so the extra UART2USB cable for debugging is not required.



   The download link provides the document, demo videos and details on MCU-Link. If you have any questions, or have difficulties to program the probe, see https://www.nxp.com/document/guide/getting-started-with-the-mcu-link:GS-MCU-LINK.

Parent topic:*Install Xplorer Toolchains*

Install Xtensa Software Tools without IDE The Xtensa Software Tools can also be installed without the use of the IDE. The installation without the IDE is useful in a command-line only Linux environment, or for better compatibility with an unsupported Linux environment.

The command-line tools package is available as a redistributable zip file that is extracted with an Xplorer IDE install. To gain access to the tools package, the IDE must be installed once in the organization. The tools package is available at: ~/xtensa/XtDevTools/downloads/RI-2023.11/tools/XtensaTools_RI_2023_11_linux.tgz..

With the tools package and the DSP Build Configuration package available from the Tensilica Tools download site, the toolchain can be set up as follows:

```
# Create Xtensa install root
mkdir -p ~/xtensa/tools
mkdir -p ~/xtensa/builds
# Set up the configuration-independent Xtensa Tool:
tar zxvf XtensaTools_RI_2023_11_linux.tgz -C ~/xtensa/tools
# Set up the configuration-specific core files:
tar zxvf nxp_rt700_RI23_11_newlib_linux_redist.tgz -C ~/xtensa/builds
# Install the Xtensa development toolchain:
cd ~/xtensa
./builds/RI-2023.11-linux/nxp_rt700_RI23_11_newlib/install \
--xtensa-tools./tools/RI-2023.11-linux/XtensaTools \
--registry ./tools/RI-2023.11-linux/XtensaTools/configg
```

Parent topic:*Install Xplorer Toolchains*

Install MCUXpresso SDK

This section list the steps to: Download MCUXpresso SDK for RT700.

Download MCUXpresso SDK for RT700 DSP enablement for RT700, including drivers, middleware libraries, and demo applications are included with the latest RT700 SDK will be available for download from https://mcuxpresso.nxp.com. If you are accessing the site for the first time, make sure to register.

Parent topic:*Install MCUXpresso SDK*

Run and Debug DSP Demo using Xplorer IDE

This section lists the steps to:

- Prepare Arm Core for 'Hello World'
- Start Xtensa Debugger Daemon
- Prepare DSP Core for 'Hello World'

Prepare Arm Core for 'Hello World' Each of the DSP demos included in the MCUXpresso SDK consists of two separate applications that run on the Arm core and DSP core. The Arm core application initializes the DSP core and executes other application-specific functionality.

To debug the 'Hello World' DSP application, you must first set up and execute the Arm application using an environment of your choice. It is required to modify the global preprocessor macro, DSP_IMAGE_COPY_TO_RAM, to control loading of the DSP binary application.

When this macro is set to '0'/FALSE, the Arm application and DSP application are independently loaded and debugged.

- Build and execute the 'Hello World' Arm demo located in: <SDK_ROOT>/boards/mimxrt700evk/dsp_examples/hello_world_usart/cm33/.

Preparing an Arm core development environment is outside the scope of this document.

Note: If you are using MCUXpresso, it is highly recommended to upgrade to latest version of the SDK and match the latest EVK board.

Parent topic:*Run and Debug DSP Demo using Xplorer IDE*

Start Xtensa Debugger Daemon To debug DSP applications on RT700, ensure that the xt-ocd daemon is running. This application runs a gdb server that the Xtensa core debugger connects to.

Go to the command-line window and change directory (cd) to xt-ocd daemon installation path. The default path on Windows is C:\Program Files (x86)\Tensilica\Xtensa OCD Daemon 14.11.

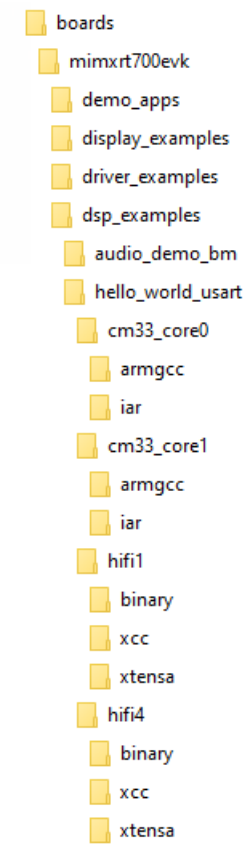
Execute the daemon with the custom topology.

```
**xt-ocd.exe -c topology.xml**
XOCD 14.11 2023-04-06 14:27:20
(c) 1999-2024 Cadence Design Systems Inc. All rights reserved.
[Debug Log 2024-02-19 22:41:53]
Loading module ”gdbstub” v2.0.0.12
Loading module ”jlink” v2.0.2.0
Using JLINK lib v.79410
Jlink USB Serial Number: 851006054
Connected to Jlink Device:
  Name:'SEGGER J-Link'
  S/N:851006054
  Firmware: J-Link V11 compiled Dec  4 2023 10:22:45
  Requested/Set TCK: 1000kHz/1000kHz
Jlink: Select pipelined SWD
SWD-DP with ID 0x6BA02477
Loading module ”jtag” v2.0.0.20
Loading module ”xtensa” v2.0.0.48
Starting thread 'GDBStub'
Opened GDB socket at port 20000
Initialize XDM driver
Warning: Warning: DAP Reset request failed! Ignoring...
```

Note: Some warning messages are expected and can be ignored. If you receive an error initializing the XDM driver, initialize and start the DSP core before debugging. For details on initializing and debugging, see Initialize DSP Core and Link DSP Profiles.

Parent topic:*Run and Debug DSP Demo using Xplorer IDE*

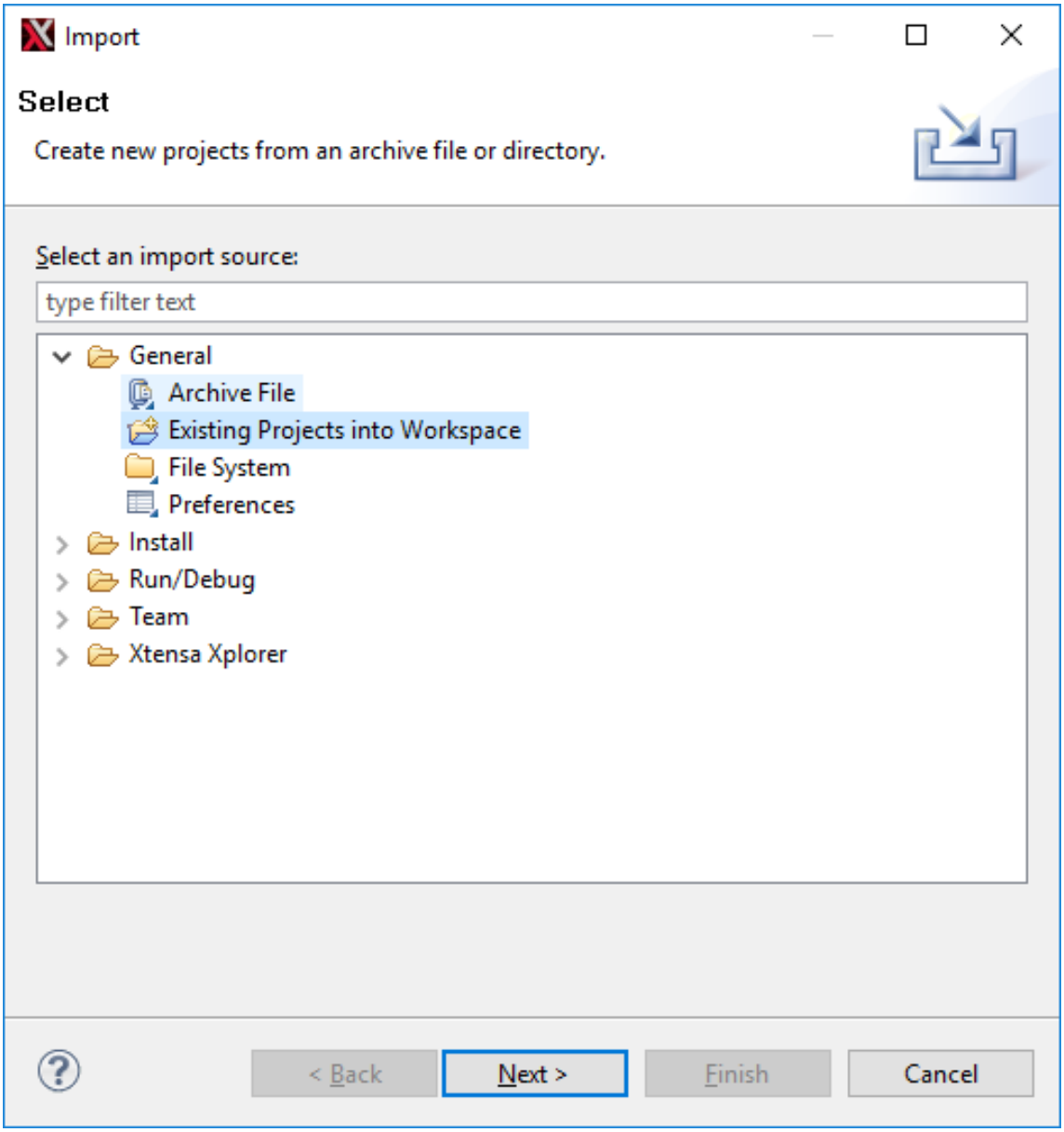
Prepare DSP Core for ‘Hello World’ The RT700 SDK provides a collection of DSP example applications located in boards/mimxrt700evk/dsp_examples/. Each DSP example has two four source directories for cm33_core0, cm33_core1, hifi1, and hifi4 cores. In this guide, HiFi4 example will be explained, but HiFi1 examples can be run with the same steps.



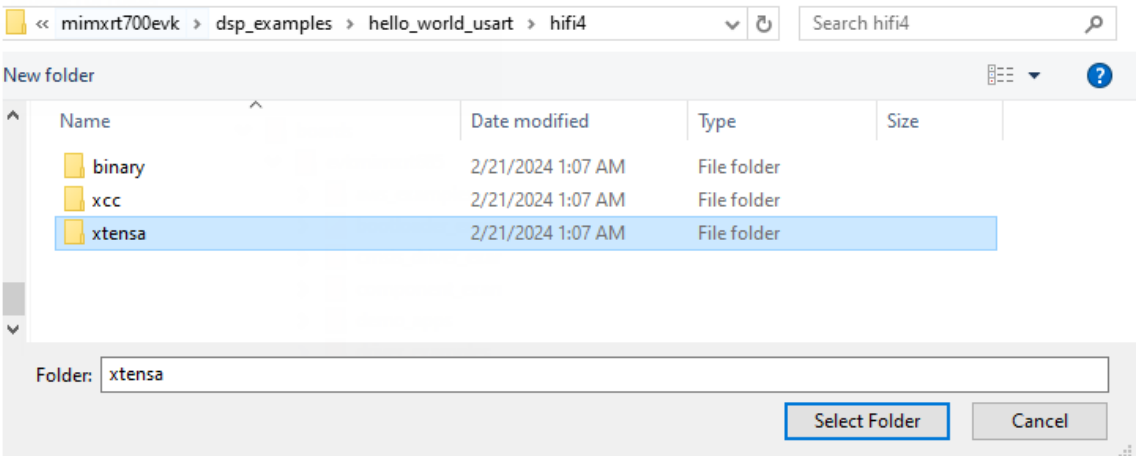
The projects for different supported toolchains are built in the above directories. For the DSP example above, the 'xcc' project builds on the command line and the 'xtensa' directory is an Xplorer IDE project.

To run the 'Hello World' demo, import the SDK sources into the Xplorer IDE.

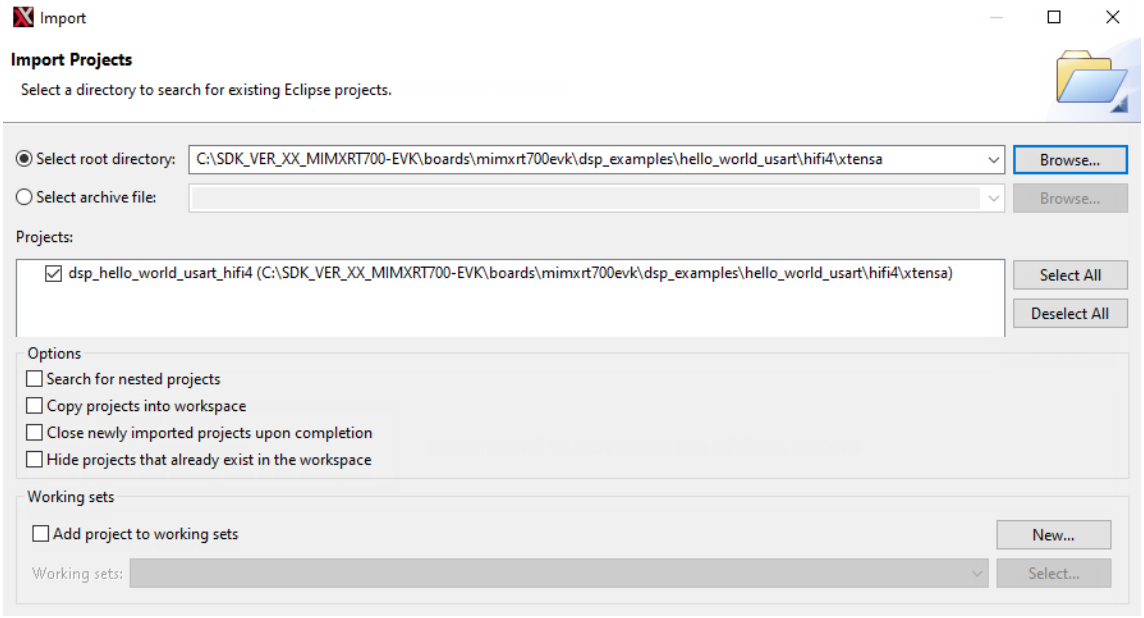
1. Select **File** > **Import** > **General** > **Existing Projects into Workspace**.



2. Select the SDK directory <SDK_ROOT>\boards\mimxrt700evk\dsp_examples\hello_world_usart\hifi4\xtensa as root directory.

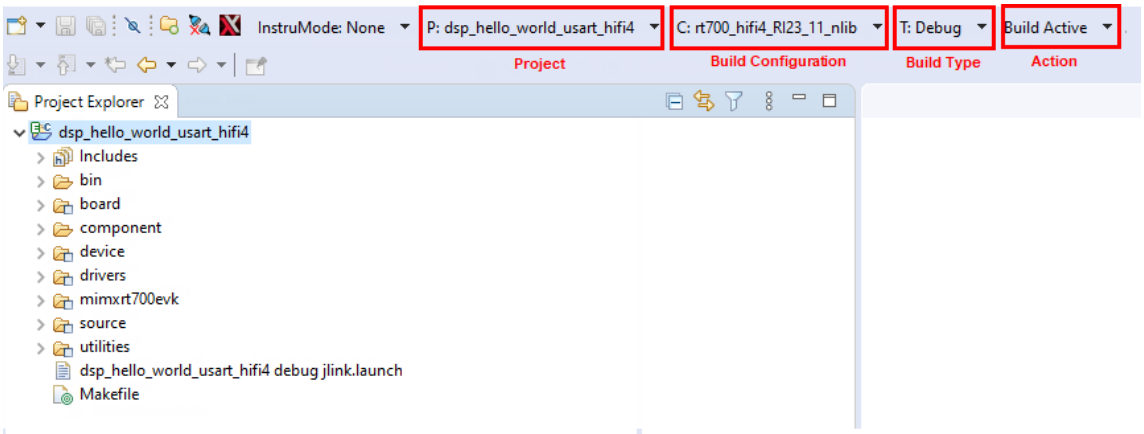


3. Leave all the other check boxes blank.

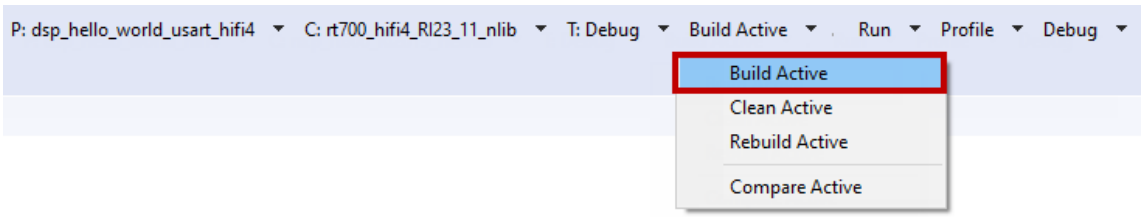


Once imported, the 'dsp_hello_world_usart_hifi4' appears in the Project Explorer.

4. To make a build selection for the project and hardware target configuration, use the drop-down buttons on the menu bar.

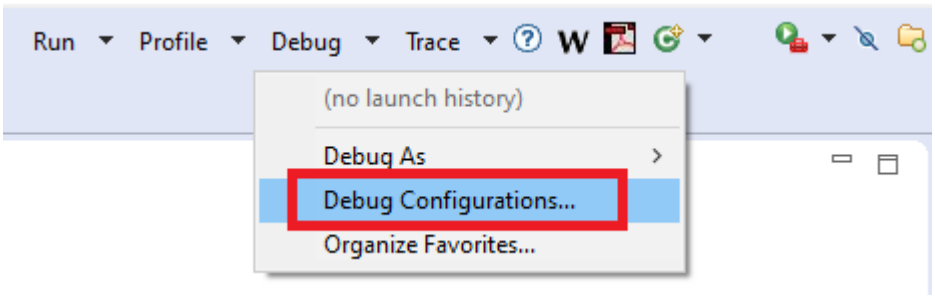


5. To build the project for debug, profile, or trace, use the action buttons on the right side of the menu bar.



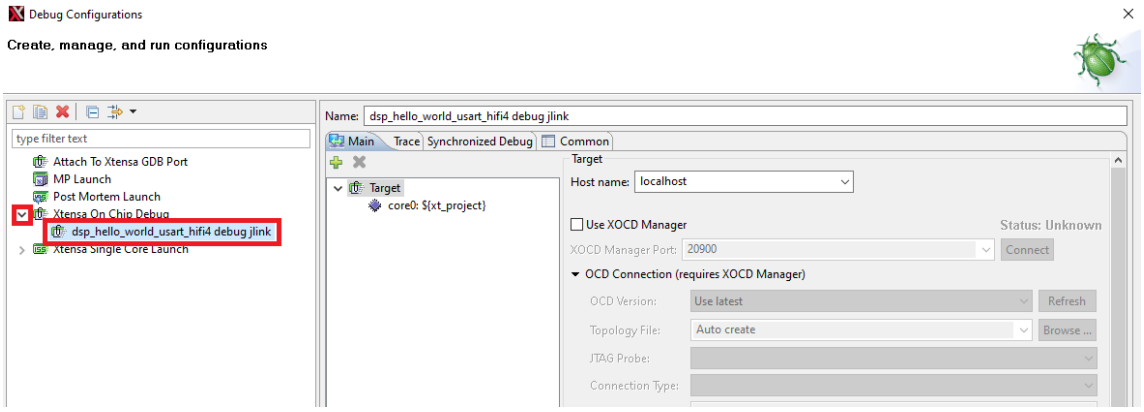
A default debug configuration is provided by the SDK project, which utilizes the on-chip debugger.

6. To choose the configuration, select the **Debug Configurations** menu item.

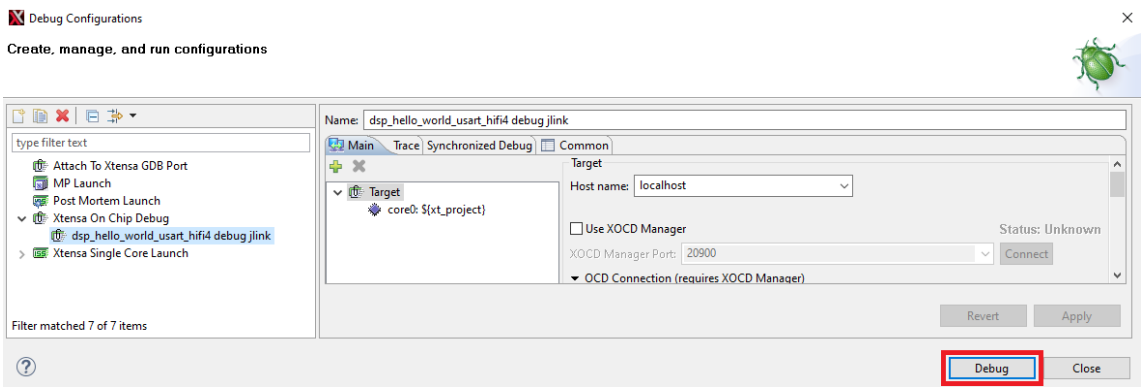


The **Debug Configurations** dialog box appears.

7. Expand the **Xtensa On Chip Debug** tree option and select '*dsp_hello_world_hifi4_debug_jlink*'.

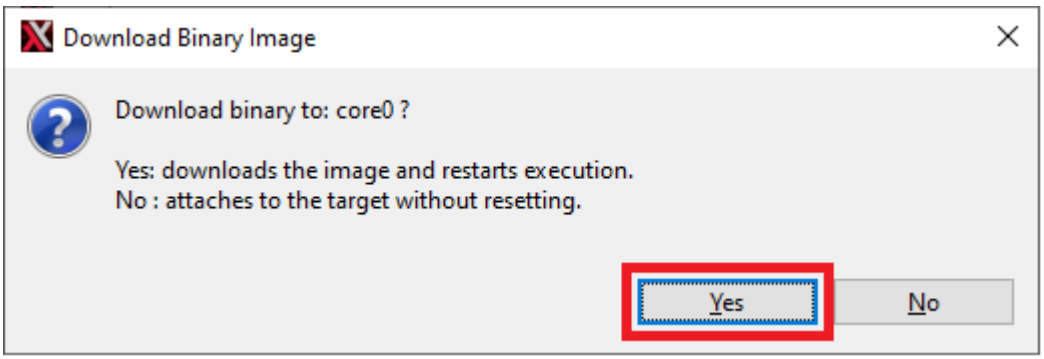


8. Click the **Debug** button. The actual debug on the chip initiates.



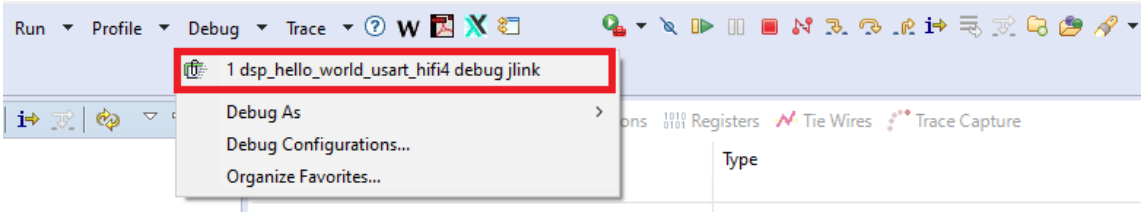
The Xplorer IDE prompts whether binaries should download to the hardware.

9. Select **Yes**.

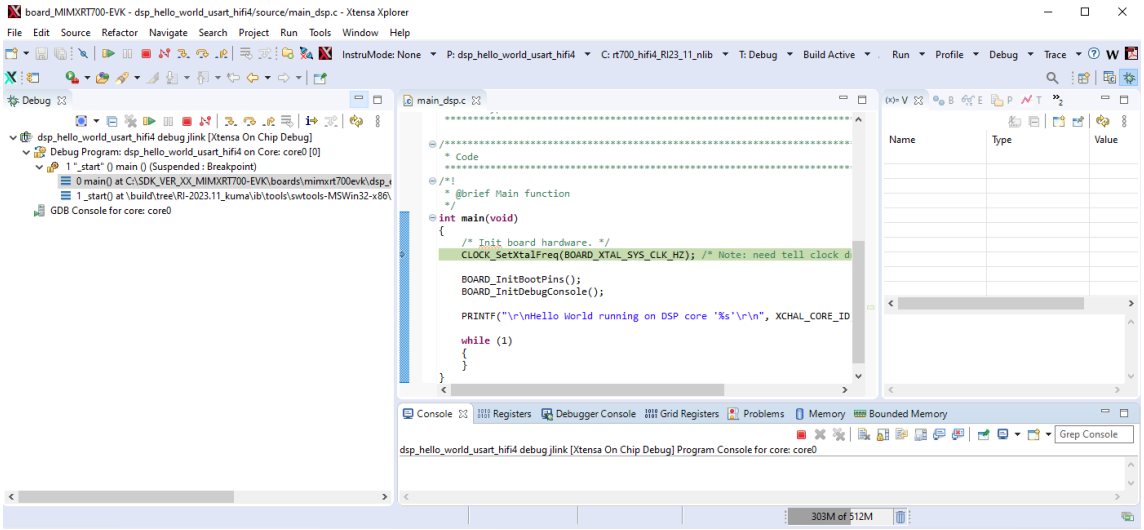


10. The Xplorer IDE transitions to the **Debug** perspective after the binary download.

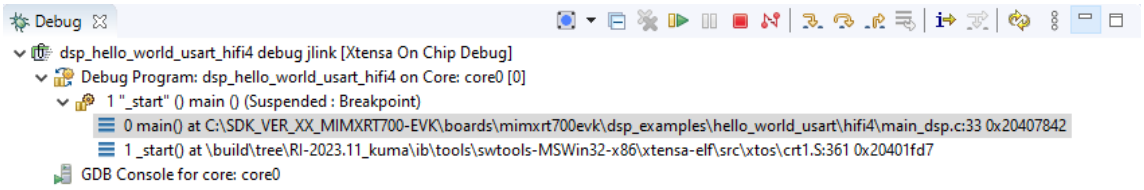
Note: After initial configuration, it is possible to select the same debug configuration as in Figure 8.



The **Debug** perspective appears as in Figure 9.



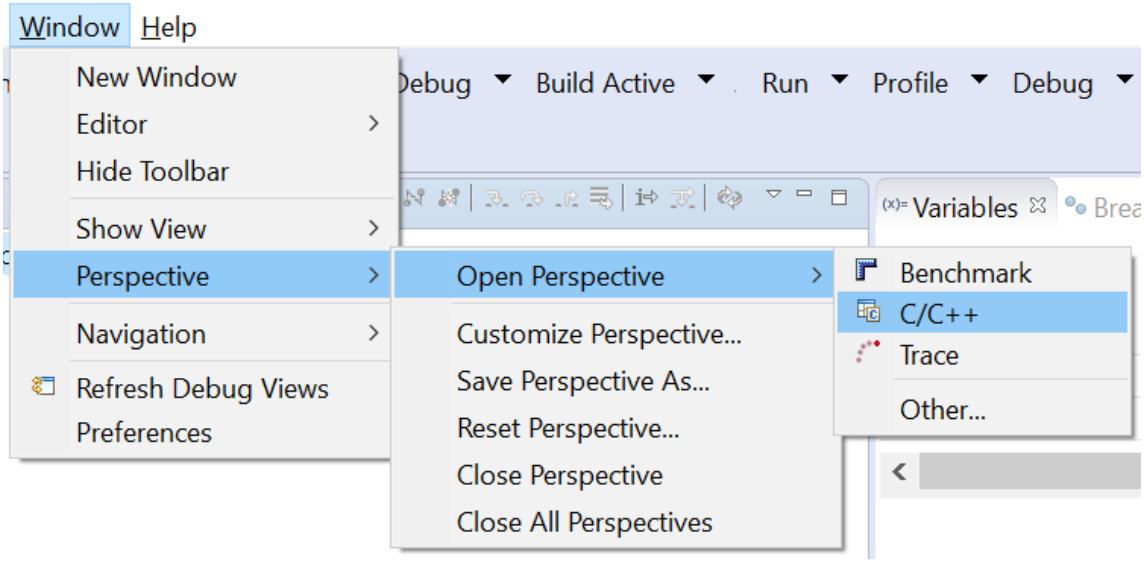
The program stops at the start of main() function. To run the program, click the **Resume** or **Stepping through** icon button as shown in Figure 10.



11. After resuming / stepping through the 'printf' statement, the following output appears in the **Console** view of the IDE.

```
Hello World running on DSP core 'rt700_hifi4_RI23_11_nlib'
```

To return to the default IDE layout after the debugging completes, select the previous code perspective.



Parent topic:*Run and Debug DSP Demo using Xplorer IDE*

1.6 Release Notes

1.6.1 MCUXpresso SDK Release Notes

Development tools

The MCUXpresso SDK was tested with following development tools. Same versions or above are recommended.

- MCUXpresso IDE, Rev. 25.06.xx
- IAR Embedded Workbench for Arm, version is 9.70.4
- Keil MDK, version is 5.43a
- MCUXpresso for VS Code v26.06
- GCC Arm Embedded Toolchain 14.2.x
- Xtensa Xplorer, version is 12.0.6
- Xtensa C Compiler, version is RJ-2026.6
- RISCV LLVM 19.1.7

Supported development systems

This release supports board and devices listed in following table. The board and devices in bold were tested in this release.

Development boards	MCU devices		
MIMXRT7EVK	MIMXRT735SGAWBR, MIMXRT758SGAWBR, MIMXRT798SGAWBR, **MIMXRT798SGFOB**, MIMXRT798SGVKB	MIMXRT735SGFOB, MIMXRT758SGFOB,	MIMXRT735SGVKB, MIMXRT758SGVKB,

MCUXpresso SDK release package

The MCUXpresso SDK release package content is aligned with the silicon subfamily it supports. This includes the boards, CMSIS, devices, middleware, and RTOS support.

Device support The device folder contains the whole software enablement available for the specific System-on-Chip (SoC) subfamily. This folder includes clock-specific implementation, device register header files, device register feature header files, and the system configuration source files. Included with the standard SoC support are folders containing peripheral drivers, toolchain support, and a standard debug console. The device-specific header files provide a direct access to the microcontroller peripheral registers. The device header file provides an overall SoC memory mapped register definition. The folder also includes the feature header file for each peripheral on the microcontroller. The toolchain folder contains the startup code and linker files for each supported toolchain. The startup code efficiently transfers the code execution to the main() function.

Board support The boards folder provides the board-specific demo applications, driver examples, and middleware examples.

Demo application and other examples The demo applications demonstrate the usage of the peripheral drivers to achieve a system level solution. Each demo application contains a readme file that describes the operation of the demo and required setup steps. The driver examples demonstrate the capabilities of the peripheral drivers. Each example implements a common use case to help demonstrate the driver functionality.

RTOS

FreeRTOS Real-time operating system for microcontrollers from Amazon

Middleware

Wireless EdgeFast Bluetooth PAL For more information, see the MCUXpresso SDK EdgeFast Bluetooth Protocol Abstraction Layer User's Guide.

Wireless EdgeFast Bluetooth PAL will be deprecated starting with SDK v26.09.00.

SDK EdgeFast v26.06.00 is classified as a Long Term Support (LTS).

corePKCS11 Reference implementation of the PKCS#11 standard for secure credential storage and cryptographic operations.

coreMQTT Lightweight MQTT client library for IoT devices from the AWS IoT Device SDK.

coreJSON Strictly conforming RFC 8259 JSON parser library from the AWS IoT Device SDK.

coreHTTP coreHTTP

backoffAlgorithm Library for calculating exponential backoff with jitter for network retry attempts.

openvg OpenVG library for devices with graphics acceleration hardware

NXP Wi-Fi The MCUXpresso SDK provides driver for NXP Wi-Fi external modules. The Wi-Fi driver is integrated with LWIP TCPIP stack and demonstrated with several network applications (iperf and AWS IoT).

For more information, see Getting Started with NXP based Wireless Modules and i.MX RT Platform Running on RTOS (document: UM11441).

Voice intelligent technology library Voice Intelligent Technology (VIT) Library provides wake word and voice command engine for voice control

VG-Lite GPU Library VGLite library for devices with VGLite graphics hardware acceleration engine

USB Type-C PD Stack See the *MCUXpresso SDK USB Type-C PD Stack User's Guide* (document MCUXSDKUSBPDUG) for more information

USB Host, Device, OTG Stack See the MCUXpresso SDK USB Stack User's Guide (document MCUXSDKUSBSUG) for more information.

Peripheral devices tested with USB Host stack

Table 1 provides a list of USB devices tested with the USB Host stack.

Device type	Device
USB HUB	BELKIN F5U233BELKIN F5U304BELKIN F5U307BELKIN F4U040UNITEK Y-2151Z-TEK ZK032AHYUNDAI HY-HB608
USB flash drive	ADATA C008 32 GBADATA S102 8 GADATA S102 16 GVerbatim STORE N GO USB Device 8 GKingston DataTraveler DT101 G2SanDisk Cruzer Blade 8 GBUnisplendour 1 GImation 2 GBV-mux 2 GBSanmina-SCI 128 MCorporate Express 1 GTOSHIBA THUHYBS-008G 8 GTranscend JF700 8 GNetac U903 16 GSSK SFD205 8 GBRex 4 GBSAMSUNG USB3.0 16GB
USB card reader/	SSK TF adapterKawau Multi Card ReaderKawau TF adapterKawau SDHC cardUSB Mouse DELL MS111-PDELL M066U0ADELL MUAVDEL8TARGUS AMU76APDELL MD56U0DELL MS111-TRAPOO M110
USB Keyboard	DELL SK8135DELL SK8115

TinyCBOR Concise Binary Object Representation (CBOR) Library

TF-M Trusted Firmware - M Library

PSA Test Suite Arm Platform Security Architecture Test Suite

SDMMC stack The SDMMC software is integrated with MCUXpresso SDK to support SD/MMC/SDIO standard specification. This also includes a host adapter layer for bare-metal/RTOS applications.

PKCS#11 The PKCS#11 standard specifies an application programming interface (API), called "Cryptoki," for devices that hold cryptographic information and perform cryptographic functions. Cryptoki follows a simple object based approach, addressing the goals of technology independence (any kind of device) and resource sharing (multiple applications accessing multiple devices), presenting to applications a common, logical view of the device called a "cryptographic token".

Multicore Multicore Software Development Kit

MCU Boot Open source MCU Bootloader.

mbedTLS mbedtls SSL/TLS library v3.x

lwIP The lwIP TCP/IP stack is pre-integrated with MCUXpresso SDK and runs on top of the MCUXpresso SDK Ethernet driver with Ethernet-capable devices/boards.

For details, see the *lwIP TCPIP Stack and MCUXpresso SDK Integration User's Guide* (document MCUXSDKLWIPUG).

lwIP is a small independent implementation of the TCP/IP protocol suite.

eIQ The package contains several example applications using the eIQ TensorFlow Lite for Microcontrollers library.

eIQ machine learning SDK containing:

- Arm CMSIS-NN library (neural network kernels optimized for Cortex-M cores)
- Inference engines:
  - TensorFlow Lite Micro
  - DeepView RT
- Example code for TensorFlow Lite Micro, Glow, and DeepView RT

LVGL LVGL Open Source Graphics Library

llhttp HTTP parser llhttp

LittleFS LittleFS filesystem stack

JPEG library JPEG library

FreeMASTER FreeMASTER communication driver for 32-bit platforms.

File systemFatfs The FatFs file system is integrated with the MCUXpresso SDK and can be used to access either the SD card or the USB memory stick when the SD card driver or the USB Mass Storage Device class implementation is used.

emWin The MCUXpresso SDK is pre-integrated with the SEGGER emWin GUI middleware. The AppWizard provides developers and designers with a flexible tool to create stunning user interface applications, without writing any code.

DSP Neural Networks DSP Neural Networks Framework based on Xtensa Neural Networks Library from Cadence Design Systems for Xtensa HiFi Audio Engines.

NatureDSP for Hifi4 Digital Signal Processing for Xtensa Hifi4 DSP Audio Engines.

NatureDSP for Hifi1 Digital Signal Processing for Xtensa Hifi1 DSP Audio Engines.

DSP Audio Streamer DSP Audio Streamer Framework based on Xtensa Audio Framework from Cadence Design Systems for Xtensa DSP Audio Engines.

DSP Codecs Codecs for Cadence DSPs

NXP PSA CRYPTO DRIVER PSA crypto driver for crypto library integration via driver wrappers

NXP ELS PKC ELS PKC crypto library

CMSIS DSP Library The MCUXpresso SDK is shipped with the standard CMSIS development pack, including the prebuilt libraries.

Release contents

Provides an overview of the MCUXpresso SDK release package contents and locations.

Deliverable	Location
Boards	INSTALL_DIR/boards
Demo Applications	INSTALL_DIR/boards/<board_name>/demo_apps
Driver Examples	INSTALL_DIR/boards/<board_name>/driver_examples
eIQ examples	INSTALL_DIR/boards/<board_name>/eiq_examples
Board Project Template for MCUXpresso IDE NPW	INSTALL_DIR/boards/<board_name>/project_template
Driver, SoC header files, extension header files and feature header files, utilities	INSTALL_DIR/devices/<device_name>
CMSIS drivers	INSTALL_DIR/devices/<device_name>/cmsis_drivers
Peripheral drivers	INSTALL_DIR/devices/<device_name>/drivers
Toolchain linker files and startup code	INSTALL_DIR/devices/<device_name>/<toolchain_nam
Utilities such as debug console	INSTALL_DIR/devices/<device_name>/utilities
Device Project Template for MCUXpresso IDE NPW	INSTALL_DIR/devices/<device_name>/project_templat
CMSIS Arm Cortex-M header files, DSP library source	INSTALL_DIR/CMSIS
Components and board device drivers	INSTALL_DIR/components
RTOS	INSTALL_DIR/rtos
Release Notes, Getting Started Document and other documents	INSTALL_DIR/docs
Tools such as shared cmake files	INSTALL_DIR/tools
Middleware	INSTALL_DIR/middleware

Known issues

This section lists the known issues, limitations, and/or workarounds.

AHBSC protection not enforced on B0 silicon which version=T2.0.32

An erratum (ERR053561) affects B0 silicon (version T2.0.32). Due to this issue, software that depends on AHBSC protection mechanisms may experience unintended behavior because the configured protection settings are not properly enforced.

To resolve the issue, follow the guidance provided in the errata document and apply the corresponding ROM patch.

Examples impacted by this erratum include applications and examples that rely on AHBSC protection features, such as:

trustzone_examples freertos_examples/freertos_mpu

These examples should be updated with the recommended workaround to ensure correct protection behavior on affected devices.

1.7 ChangeLog

1.7.1 MCUXpresso SDK Changelog

Board Support Files

board [25.12.00]

- Improvement
  - Updated BOARD_ConfigMPU to support configuring SRAM as write-through by using CACHE_MODE_WRITE_THROUGH.

[25.09.00]

- Improvement
  - Added BOARD_SetDeepSleepPinConfig and BOARD_RestoreDeepSleepPinConfig APIs.

[25.06.00]

- Initial version

clock_config [25.06.00]

- Initial version

pin_mux [25.06.00]

- Initial version

ACMP

[2.5.0]

- New Feature
  - Supported the plateforms which don't have continuous mode.

[2.4.0]

- New Feature
  - Supported the plateforms which don't have continuous mode.

[2.3.0]

- Improvements
  - Expose C0 register FILTER_CNT bitfield and FPR bitfield to the user.

[2.2.0]

- Improvements
  - Updated feature macros for roundrobin mode, window mode, filter mode, and 3V domain removes.

[2.1.0]

- New Feature
  - Supported the plateforms which don't have hysteresis mode.

[2.0.6]

- Bug Fixes
  - Fixed the wrong comments, the DAC value should range from 0 to 255.

[2.0.5]

- Bug Fixes
  - Fixed the out-of-bounds error of Coverity caused by missing an assert sentence to avoid the return value of ACMP_GetInstance() exceeding the array bounds.
  - Fixed the violations of MISRA C-2012 rules:
    - Rule 10.1, 14.4, 16.4, 17.7.

[2.0.4]

- Bug Fixes
  - Avoided changing w1c bit in ACMP_SetRoundRobinPreState().

[2.0.3]

- New Features
  - Added feature functions for usage of different power domains(1.8 V and 3 V). These functions are first enabled in ULP1. They are about:
    - ACMP_EnableLinkToDAC()
    - ACMP_SetDiscreteModeConfig()
    - ACMP_GetDefaultDiscreteModeConfig()

[2.0.2]

- Other Changes
  - Changed coding style of peripheral base address from “s_acmpBases” to “s_acmpBase”.

[2.0.1]

- Bug Fixes
  - Fixed bug regarding the function “ACMP_SetRoundRobinConfig”. It will not continue execution but returns directly after disabling round robin mode.

CACHE64

[2.0.14]

- Bug Fixes
  - Fixed boundary check in CACHE64_GetInstanceByAddr() to correctly include the last address of the memory range.

[2.0.13]

- Bug Fixes
  - Fix issue in CACHE64_InvalidateCacheByRange(), CACHE64_CleanCacheByRange() and CACHE64_CleanInvalidateCacheByRange() functions to avoid cleaning/invalidation less cache lines than expected.

[2.0.12]

- Improvements
  - Add memory barrier when enabling/disabling cache.

[2.0.11]

- Bug Fixes
  - Fixed CERT INT30-C violations: check and guarantee address plus size is equal or smaller than UINT32_MAX.

[2.0.10]

- Improvements
  - Updated CACHE64_InvalidateCacheByRange(), CACHE64_CleanCacheByRange() and CACHE64_CleanInvalidateCacheByRange() to support some platforms that multiple regions in the memory map are remapped to create a continuous address space.

[2.0.9]

- Improvements
  - Removed assert(false) in CACHE64_GetInstanceByAddr.

[2.0.8]

- Improvements
  - Updated function CACHE64_GetInstanceByAddr() to support some devices that provide alias of cacheable memory section.

[2.0.7]

- Improvements
  - Check input parameter “size_byte” must be larger than 0.

[2.0.6]

- Bug Fixes
  - Fixed overflow for CACHE64_GetInstanceByAddr()/CACHE64_CleanCacheByRange()/CACHE64_Invalid
    APIs.

[2.0.5]

- Improvement
  - Made use of FSL_FEATURE_CACHE64_CTRL_HAS_NO_WRITE_BUF feature

[2.0.4]

- Improvement
  - Disable cache policy feature on SoC without CACHE64_POLSEL IP.
- Bug Fixes
  - Fixed doxygen issue.

[2.0.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.3.

[2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3, 10.4 and 14.4.
  - Fixed doxygen issue.

[2.0.1]

- Improvements
  - Moved CLCR register configuration out of the while loop, it's unnecessary to repeat this operation.

[2.0.0]

- Initial version.

CACHE XCACHE

[2.0.6]

- Improvements
  - Add assert in XCACHE_GetInstanceByAddr() to check out-of-range address.

[2.0.5]

- Bug Fixes
  - Fixed boundary check in XCACHE_GetInstanceByAddr() to correctly include the last address of the memory range.

[2.0.4]

- Improvements
  - Add memory barrier when enabling/disabling cache.

[2.0.3]

- Bug Fixes
  - Fixed CERT INT30-C violations.

[2.0.2]

- Bug Fixes
  - Updated XCACHE_InvalidateCacheByRange(), XCACHE_CleanCacheByRange(), XCACHE_CleanInvalidateCacheByRange() in case of startAddr equal to endAddr.

[2.0.1]

- Improvements
  - Check input parameter “size_byte” must be larger than 0.

[2.0.0]

- Initial version.

CDOG

[2.1.3]

- Re-design multiple instance IRQs and Clocks
- Add fix for RESTART command errata

[2.1.2]

- Support multiple IRQs
- Fix default CONTROL values

[2.1.1]

- Remove bit CONTROL[CONTROL_CTRL].

[2.1.0]

- Rename CWT to CDOG.

[2.0.2]

- Fix MISRA-2012 issues.

[2.0.1]

- Fix doxygen issues.

[2.0.0]

- Initial version.

CLOCK

[2.5.1]

- Improvement
  - Added a check for the temporary parameter to prevent potential wrapping issues.

[2.5.0]

- New features
  - Updated CLOCK_EnableFroClkFreqCloseLoop to support timeout mechanism.
  - Added new API CLOCK_FroTuneOnce to allow frequency measurement in open loop mode.

[2.4.2]

- Improvement
  - Updated CLOCK_GetSenseAudioClkFreq for compute domain to support silicon B0.
  - Fixed MISRA issues.

[2.4.1]

- Bug fixes
  - Added kCLOCK_Pvts0.
  - Fixed bugs in CLOCK_GetWakeClk32KFreq. Reserved CLKCTL3->WAKE32KCLKDIV.

[2.4.0]

- New features
  - Added return value for CLOCK_InitMainPfd CLOCK_InitAudioPfd.
  - Updated FRO max supported frequency.
  - Added fast startup configuration for FRO.

[2.3.1]

- Bug fixes
  - Fixed bugs in CLOCK_ClearFroFlags.
  - Fixed bugs in clock_attach_id_t.

[2.3.0]

- Bug fixes
  - Added clock array for SDADC, CDOG.
  - Added FRO related APIs.

[2.2.2]

- Bug fixes
  - Fixed GPIO_CLOCKS array for GPIO_ALIAS.

[2.2.1]

- Bug fixes
  - Fixed bugs in clock_attach_id_t.
  - Fixed FRO frequency calcualtion.

[2.2.0]

- Bug fixes
  - Fixed UTICK0 clock sources.

[2.1.0]

- New features
  - Added CLOCK_GetCoreSysClkFreq API.
- Bug fixes
  - Fixed bugs in CLOCK_GetFCClkFreq and clock_attach_id_t.

[2.0.0]

- Initial version.

COMMON

[2.6.7]

- Improvements
  - EnableIRQ/DisableIRQ: Use IRQSTEER_GetInstIdxByIRQ() to determine IRQSTEER instance index instead of simple division. Supports multiple IRQSTEER instances with different interrupt counts.

[2.6.6]

- Improvements
  - Support Cortex M55.

[2.6.5]

- Bug Fixes
  - Fixed the issue that the function SDK_DelayAtLeastUs may underflow and cause a long delay when the input parameter is small and the system clock is low.

[2.6.4]

- New Features
  - Add macros to tag deprecated features and experimental features.

[2.6.3]

- New Features
  - Added bit mask inversion macros to avoid type promotion.
  - Added register operation macros.
- Improvements
  - Make function MSDK_EnableCpuCycleCounter compatible with CMSIS-5 and CMSIS-6.
- Bug Fixes
  - Fixed build issue of CMSIS PACK BSP example caused by CMSIS 6.1 issue.

[2.6.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule for implicit conversions in boolean contexts

[2.6.1]

- Improvements
  - Support Cortex M23.

[2.6.0]

- Bug Fixes
  - Fix CERT-C violations.

[2.5.0]

- New Features
  - Added new APIs InitCriticalSectionMeasurementContext, DisableGlobalIRQEx and EnableGlobalIRQEx so that user can measure the execution time of the protected sections.

[2.4.3]

- Improvements
  - Enable irqs that mount under irqsteer interrupt extender.

[2.4.2]

- Improvements
  - Add the macros to convert peripheral address to secure address or non-secure address.

[2.4.1]

- Improvements
  - Improve for the macro redefinition error when integrated with zephyr.

[2.4.0]

- New Features
  - Added EnableIRQWithPriority, IRQ_SetPriority, and IRQ_ClearPendingIRQ for ARM.
  - Added MSDK_EnableCpuCycleCounter, MSDK_GetCpuCycleCount for ARM.

[2.3.3]

- New Features
  - Added NETC into status group.

[2.3.2]

- Improvements
  - Make driver aarch64 compatible

[2.3.1]

- Bug Fixes
  - Fixed MAKE_VERSION overflow on 16-bit platforms.

[2.3.0]

- Improvements
  - Split the driver to common part and CPU architecture related part.

[2.2.10]

- Bug Fixes
  - Fixed the ATOMIC macros build error in cpp files.

[2.2.9]

- Bug Fixes
  - Fixed MISRA C-2012 issue, 5.6, 5.8, 8.4, 8.5, 8.6, 10.1, 10.4, 17.7, 21.3.
  - Fixed SDK_Malloc issue that not allocate memory with required size.

[2.2.8]

- Improvements
  - Included stddef.h header file for MDK tool chain.
- New Features:
  - Added atomic modification macros.

[2.2.7]

- Other Change
  - Added MECC status group definition.

[2.2.6]

- Other Change
  - Added more status group definition.
- Bug Fixes
  - Undef __VECTOR_TABLE to avoid duplicate definition in cmsis_clang.h

[2.2.5]

- Bug Fixes
  - Fixed MISRA C-2012 rule-15.5.

[2.2.4]

- Bug Fixes
  - Fixed MISRA C-2012 rule-10.4.

[2.2.3]

- New Features
  - Provided better accuracy of SDK_DelayAtLeastUs with DWT, use macro SDK_DELAY_USE_DWT to enable this feature.
  - Modified the Cortex-M7 delay count divisor based on latest tests on RT series boards, this setting lets result be closer to actual delay time.

[2.2.2]

- New Features
  - Added include RTE_Components.h for CMSIS pack RTE.

[2.2.1]

- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 3.1, 10.1, 10.3, 10.4, 11.6, 11.9.

[2.2.0]

- New Features
  - Moved SDK_DelayAtLeastUs function from clock driver to common driver.

[2.1.4]

- New Features
  - Added OTFAD into status group.

[2.1.3]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed the rule: rule-10.3.

[2.1.2]

- Improvements
  - Add SUPPRESS_FALL_THROUGH_WARNING() macro for the usage of suppressing fallthrough warning.

[2.1.1]

- Bug Fixes
  - Deleted and optimized repeated macro.

[2.1.0]

- New Features
  - Added IRQ operation for XCC toolchain.
  - Added group IDs for newly supported drivers.

[2.0.2]

- Bug Fixes
  - MISRA C-2012 issue fixed.
    - Fixed the rule: rule-10.4.

[2.0.1]

- Improvements
  - Removed the implementation of LPC8XX Enable/DisableDeepSleepIRQ() function.
  - Added new feature macro switch "FSL_FEATURE_HAS_NO_NONCACHEABLE_SECTION" for specific SoCs which have no noncacheable sections, that helps avoid an unnecessary complex in link file and the startup file.
  - Updated the align(x) to **attribute**(aligned(x)) to support MDK v6 armclang compiler.

[2.0.0]

- Initial version.

CRC

[2.1.0]

- Bug fix:
  - Choosing CRC clocks from CRC clock array according to instance instead of hardcoded value.

[2.0.5]

- Bug fix:
  - Fix CERT-C issue with boolean-to-unsigned integer conversion.

[2.0.4]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.0.3]

- Bug fix:
  - Fix MISRA issues.

[2.0.2]

- Bug fix:
  - Fix MISRA issues.

[2.0.1]

- Bug fix:
  - DATA and DATALL macro definition moved from header file to source file.

[2.0.0]

- Initial version.

CTIMER

[2.3.6]

- Bug Fixes
  - Fix CERT INT30-C issue.

[2.3.5]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.7

[2.3.4]

- Bug Fixes
  - Fixed ERRATA ERR053024 CTIMER will enter interrupt twice when function clock much slower than bus clock.

[2.3.3]

- Bug Fixes
  - Fix CERT INT30-C INT31-C issue.
  - Make API CTIMER_SetupPwm and CTIMER_UpdatePwmDutycycle return fail if pulse width register overflow.

[2.3.2]

- Bug Fixes
  - Clear unexpected DMA request generated by RESET_PeripheralReset in API CTIMER_Init to avoid trigger DMA by mistake.

[2.3.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.7 and 12.2.

[2.3.0]

- Improvements
  - Added the CTIMER_SetPrescale(), CTIMER_GetCaptureValue(), CTIMER_EnableResetMatchChannel(), CTIMER_EnableStopMatchChannel(), CTIMER_EnableRisingEdgeCapture(), CTIMER_EnableFallingEdgeCapture(), CTIMER_SetShadowValue(),APIs Interface to reduce code complexity.

[2.2.2]

- Bug Fixes
  - Fixed SetupPwm() API only can use match 3 as period channel issue.

[2.2.1]

- Bug Fixes
  - Fixed use specified channel to setting the PWM period in SetupPwmPeriod() API.
  - Fixed Coverity Out-of-bounds issue.

[2.2.0]

- Improvements
  - Updated three API Interface to support Users to flexibly configure the PWM period and PWM output.
- Bug Fixes
  - MISRA C-2012 issue fixed: rule 8.4.

[2.1.0]

- Improvements
  - Added the CTIMER_GetOutputMatchStatus() API Interface.
  - Added feature macro for FSL_FEATURE_CTIMER_HAS_NO_CCR_CAP2 and FSL_FEATURE_CTIMER_HAS_NO_IR_CR2INT.

[2.0.3]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.3, 10.4, 10.6, 10.7 and 11.9.

[2.0.2]

- New Features
  - Added new API "CTIMER_GetTimerCountValue" to get the current timer count value.
  - Added a control macro to enable/disable the RESET and CLOCK code in current driver.
  - Added a new feature macro to update the API of CTimer driver for lpc8n04.

[2.0.1]

- Improvements
  - API Interface Change
    - Changed API interface by adding CTIMER_SetupPwmPeriod API and CTIMER_UpdatePwmPulsePeriod API, which both can set up the right PWM with high resolution.

[2.0.0]

- Initial version.

EDMA

[2.10.14]

- Improvements
  - Applied ERRATA 52315.

[2.10.13]

- Improvements
  - Introduced the version-neutral name driver.edma_unified for the unified eDMA driver (which serves the eDMA3/eDMA4/eDMA5 IP versions), renamed the driver directory to drivers/edma_unified, and set FSL_COMPONENT_ID to platform.drivers.edma_unified. The previous driver.edma4 Kconfig component and the FSL_EDMA_DRIVER_EDMA4 macro are retained as deprecated aliases, so existing projects (including the shipped eDMA examples) keep building without changes; new projects should use driver.edma_unified / FSL_EDMA_DRIVER_UNIFIED. The C API, edma_handle_t, registers, and the IP-specific edma4_*/edma5_* TCD identifiers are unchanged.

[2.10.12]

- Bug Fixes
  - Fixed dynamic scatter/gather mid-flight submit on eDMA4/eDMA5: resolve the TCD link in software under a quiescent fence, since these IPs removed the CSR[ESG] read-back coherency model.

[2.10.11]

- Bug Fixes
  - Fixed CERT INT31-C and CERT INT30-C violations in edma4 driver.

[2.10.10]

- Bug Fixes
  - Fixed $\mathrm{EDMA_StartTransfer}$ could read an incorrect address when checking the channelmux status, potentially causing a hard fault issue.

[2.10.9]

- Bug Fixes
  - Add new api EDMA_TcdInit to avoid destroying code logic by reordering blocks in the toolchain.

[2.10.8]

- Bug Fixes
  - Fixed coverity issues with CERT INT30-C, CERT INT31-C compliance.
  - Fixed incorrect enabling of preemption capability issue.

[2.10.7]

- Improvements
  - Add condition to fix build warnings(array subscript 4 is above array bounds of 'edma_handle_t *[4][64]')
- Bug Fixes
  - Fixed the EDMA header index retrieval error caused by done bit calculation mistake issue.

[2.10.6]

- Improvements
  - Add macro FSL_FEATURE_EDMA_HAS_EDMA_TCD_CLOCK_ENABLE to enable tcd clocks in EDMA_Init function.

[2.10.5]

- Bug Fixes
  - Fixed memory convert would convert NULL as zero address issue.

[2.10.4]

- Improvements
  - Add new MP register macros to ensure compatibility with different devices.
  - Add macro DMA_CHANNEL_ARRAY_STEPn to adapt to complex addressing of edma tcd registers.

[2.10.3]

- Bug Fixes
  - Clear interrupt status flags in EDMA_CreateHandle to avoid triggering interrupt by mistake.

[2.10.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3.

[2.10.1]

- Bug Fixes
  - Fixed EDMA_GetRemainingMajorLoopCount may return wrong value issue.
  - Fixed violations of the MISRA C-2012 rules 13.5, 10.4.

[2.10.0]

- Improvements
  - Modify the structures edma_core_mp_t, edma_core_channel_t, edma_core_tcd_t to adapt to edma5.
  - Add TCD register macro to facilitate confirmation of tcd type.
  - Modfiy the mask macro to a fixed value.
  - Add EDMA_TCD_TYPE macro to determine edma tcd type.
  - Add extension API to the following API to determine edma tcd type.
    - EDMA_ConfigChannelSoftwareTCD -> EDMA_ConfigChannelSoftwareTCDExt
    - EDMA_TcdReset -> EDMA_TcdResetExt
    - EDMA_TcdSetTransferConfig -> EDMA_TcdSetTransferConfigExt
    - EDMA_TcdSetMinorOffsetConfig -> EDMA_TcdSetMinorOffsetConfigExt
    - EDMA_TcdSetChannelLink -> EDMA_TcdSetChannelLinkExt
    - EDMA_TcdSetBandWidth -> EDMA_TcdSetBandWidthExt
    - EDMA_TcdSetModulo -> EDMA_TcdSetModuloExt
    - EDMA_TcdEnableAutoStopRequest -> EDMA_TcdEnableAutoStopRequestExt
    - EDMA_TcdEnableInterrupts -> EDMA_TcdEnableInterruptsExt
    - EDMA_TcdDisableInterrupts -> EDMA_TcdDisableInterruptsExt
    - EDMA_TcdSetMajorOffsetConfig -> EDMA_TcdSetMajorOffsetConfigExt

[2.9.2]

- Improvements
  - Remove tcd alignment check in API that is low level and does not necessarily use scather/gather mode.

[2.9.1]

- Bug Fixes
  - Deinit channel request source before set channel mux.

[2.9.0]

- Improvements
  - Release peripheral from reset if necessary in init function.
- Bug Fixes
  - Fixed the variable type definition error issue.
  - Fixed doxygen warning.
  - Fixed violations of MISRA C-2012 rule 18.1.

[2.8.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3

[2.8.0]

- Improvements
  - Added feature FSL_FEATURE_EDMA_HAS_NO_CH_SBR_SEC to separate DMA without SEC bitfield.

[2.7.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3, 10.4, 11.6, 11.8, 14.3,.

[2.7.0]

- Improvements
  - Use more accurate DMA instance based feature macros.
- New Features
  - Add new APIs EDMA_PrepareTransferTCD and EDMA_SubmitTransferTCD, which support EDMA transfer using TCD.

[2.6.0]

- Improvements
  - Modify the type of parameter channelRequestSource from dma_request_source_t to int32_t in the EDMA_SetChannelMux.

[2.5.3]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3, 10.4, 11.6, 20.7, 12.2, 20.9, 5.3, 10.8, 8.4, 9.3.

[2.5.2]

- Improvements
  - Applied ERRATA 51327.

[2.5.1]

- Bug Fixes
  - Fixed the EDMA_ResetChannel function cannot reset channel DONE/ERROR status.

[2.5.0]

- Improvements
  - Added feature FSL_FEATURE_EDMA_HAS_NO_SBR_ATTR_BIT to separate DMA without ATTR bitfield.
  - Added api EDMA_GetChannelSystemBusInformation to gets the channel identification and attribute information on the system bus interface.
- Bug Fixes
  - Fixed the ESG bit not set in scatter gather mode issue.
  - Fixed the DREQ bit configuration missed in single transfer issue.
  - Cleared the interrupt status before invoke callback to avoid miss interrupt issue.
  - Removed disableRequestAfterMajorLoopComplete from edma_transfer_config_t structure as driver will handle it.
  - Fixed the channel mux configuration not compatible issue.
  - Fixed the out of bound access in function EDMA_DriverIRQHandler.

[2.4.4]

- Bug Fixes
  - Fixed comments by replacing STCD with TCD
  - Fixed the TCD overwrite issue when submit transfer request in the callback if there is a active TCD in hardware.

[2.4.3]

- Improvements
  - Added FSL_FEATURE_MEMORY_HAS_ADDRESS_OFFSET to convert the address between system mapped address and dma quick access address.
- Bug Fixes
  - Fixed the wrong tcd done count calculated in first TCD interrupt for the non scatter gather case.

[2.4.2]

- Bug Fixes
  - Fixed the wrong tcd done count calculated in first TCD interrupt by correct the initial value of the header.
  - Fixed violations of MISRA C-2012 rule 10.3, 10.4.

[2.4.1]

- Bug Fixes
  - Added clear CITER and BITER registers in EDMA_AbortTransfer to make sure the TCD registers in a correct state for next calling of EDMA_SubmitTransfer.
  - Removed the clear DONE status for ESG not enabled case to aovid DONE bit cleared unexpectedly.

[2.4.0]

- Improvements
  - Added api EDMA_EnableContinuousChannelLinkMode to support continuous link mode.
  - Added apis EDMA_SetMajorOffsetConfig/EDMA_TcdSetMajorOffsetConfig to support major loop address offset feature.
  - Added api EDMA_EnableChannelMinorLoopMapping for minor loop offset feature.
  - Removed the reduntant IRQ Handler in edma driver.

[2.3.2]

- Improvements
  - Fixed HIS ccm issue in function EDMA_PrepareTransferConfig.
  - Fixed violations of MISRA C-2012 rule 11.6, 10.7, 10.3, 18.1.
- Bug Fixes
  - Added ACTIVE & BITER & CITER bitfields to determine the channel status to fixed the issue of the transfer request cannot submit by function EDMA_SubmitTransfer when channel is idle.

[2.3.1]

- Improvements
  - Added source/destination address alignment check.
  - Added driver IRQ handler support for multi DMA instance in one SOC.

[2.3.0]

- Improvements
  - Added new api EDMA_PrepareTransferConfig to allow different configurations of width and offset.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4, 10.1.

- Fixed the Coverity issue regarding out-of-bounds write.

[2.2.0]

- Improvements
  - Added peripheral-to-peripheral support in EDMA driver.

[2.1.9]

- Bug Fixes
  - Fixed MISRA issue: Rule 10.7 and 10.8 in function EDMA_DisableChannelInterrupts and EDMA_SubmitTransfer.
  - Fixed MISRA issue: Rule 10.7 in function EDMA_EnableAsyncRequest.

[2.1.8]

- Bug Fixes
  - Fixed incorrect channel preemption base address used in EDMA_SetChannelPreemptionConfig API which causes incorrect configuration of the channel preemption register.

[2.1.7]

- Bug Fixes
  - Fixed incorrect transfer size setting.
    - Added 8 bytes transfer configuration and feature for RT series;
    - Added feature to support 16 bytes transfer for Kinetis.
  - Fixed the issue that EDMA_HandleIRQ would go to incorrect branch when TCD was not used and callback function not registered.

[2.1.6]

- Bug Fixes
  - Fixed KW3X MISRA Issue.
    - Rule 14.4, 10.8, 10.4, 10.7, 10.1, 10.3, 13.5, and 13.2.
- Improvements
  - Cleared the IRQ handler unavailable for specific platform with macro FSL_FEATURE_EDMA_MODULE_CHANNEL_IRQ_ENTRY_SHARED_OFFSET.

[2.1.5]

- Improvements
  - Improved EDMA IRQ handler to support half interrupt feature.

[2.1.4]

- Bug Fixes
  - Cleared enabled request, status during EDMA_Init for the case that EDMA is halted before reinitialization.

[2.1.3]

- Bug Fixes
  - Added clear DONE bit in IRQ handler to avoid overwrite TCD issue.
  - Optimized above solution for the case that transfer request occurs in callback.

[2.1.2]

- Improvements
  - Added interface to get next TCD address.
  - Added interface to get the unused TCD number.

[2.1.1]

- Improvements
  - Added documentation for eDMA data flow when scatter/gather is implemented for the EDMA_HandleIRQ API.
  - Updated and corrected some related comments in the EDMA_HandleIRQ API and edma_handle_t struct.

[2.1.0]

- Improvements
  - Changed the EDMA_GetRemainingBytes API into EDMA_GetRemainingMajorLoopCount due to eDMA IP limitation (see API comments/note for further details).

[2.0.5]

- Improvements
  - Added pubweak DriverIRQHandler for K32H844P (16 channels shared).

[2.0.4]

- Improvements
  - Added support for SoCs with multiple eDMA instances.
  - Added pubweak DriverIRQHandler for KL28T DMA1 and MCIMX7U5_M4.

[2.0.3]

- Bug Fixes
  - Fixed the incorrect pubweak IRQHandler name issue, which caused re-definition build errors when client set his/her own IRQHandler, by changing the 32-channel IRQHandler name to DriverIRQHandler.

[2.0.2]

- Bug Fixes
  - Fixed incorrect minorLoopBytes type definition in _edma_transfer_config struct, and defined minorLoopBytes as uint32_t instead of uint16_t.

[2.0.1]

- Bug Fixes
  - Fixed the eDMA callback issue (which did not check valid status) in EDMA_HandleIRQ API.

[2.0.0]

- Initial version.

EZHV

[2.1.2]

- Improvements.
  - Fixed some MISRA issues.

[2.1.1]

- Improvements.
  - Move macros from source file to head file.

[2.1.0]

- Improvements.
  - Added EZHV_SetCallback function parameters.
  - Added new APIs to get EZHV core status.

[2.0.0]

- Initial version.

FLEXIO

[2.3.2]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.3.1]

- New Features
  - Added common IRQ handler entry FLEXIO_CommonDriverIRQHandler.

[2.3.0]

- Improvements
  - Supported platforms which don't have DOZE mode control.
  - Added more pin control functions.

[2.2.3]

- Improvements
  - Adapter the FLEXIO driver to platforms which don't have system level interrupt controller, such as NVIC.

[2.2.2]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.2.1]

- Improvements
  - Added doxygen index parameter comment in FLEXIO_SetClockMode.

[2.2.0]

- New Features
  - Added new APIs to support FlexIO pin register.

[2.1.0]

- Improvements
  - Added API FLEXIO_SetClockMode to set flexio channel counter and source clock.

[2.0.4]

- Bug Fixes
  - Fixed MISRA 8.4 issues.

[2.0.3]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.0.2]

- Improvements
  - Split FLEXIO component which combines all flexio/flexio_uart/flexio_i2c/flexio_i2s drivers into several components: FlexIO component, flexio_uart component, flexio_i2c_master component, and flexio_i2s component.
- Bug Fixes

- Fixed MISRA issues
  - Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

[2.0.1]

- Bug Fixes
  - Fixed the dozen mode configuration error in FLEXIO_Init API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.

FLEXIO_CAMERA

[2.2.1]

- Bug Fixes
  - Fixed INT30-C INT31-C issues.
  - Set SHIFTCFG[LATST]=1 on devices where the bit is available, so each shifter stores its post-shift state and captures the final byte of the chain.

[2.2.0]

- Improvements
  - Supported platforms which don't have DOZE mode control.

[2.1.3]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.1.2]

- Improvements
  - Unified component full name to FLEXIO CAMERA (EDMA) driver.
- Bug Fixes
  - Fixed MISRA issues
    - Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

[2.1.1]

- Bug Fixes
  - The following modifications support FlexIO using multiple instances:
    - Removed FLEXIO_Reset API in module Init APIs.
    - Updated module Deinit APIs to reset the shifter/timer configuration instead of disabling module and clock.
    - Updated module Enable APIs to only support enable operation.

[2.1.0]

- New Features
  - Added Transfer prefix in transactional APIs.

FLEXIO_I2C

[2.6.3]

- Bug Fixes
  - Fixed MISRA 10.4, 10.7 and 14.4 issues.

[2.6.2]

- Improvements
  - Added timeout for while loop in FLEXIO_I2C_MasterTransferBlocking().
- Bug Fixes
  - Fixed build issues related to I2C_RETRY_TIMES.

[2.6.1]

- Bug Fixes
  - Fixed coverity issues

[2.6.0]

- Improvements
  - Supported platforms which don't have DOZE mode control.

[2.5.1]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.5.0]

- Improvements
  - Split some functions, fixed CCM problem in file fsl_flexio_i2c_master.c.

[2.4.0]

- Improvements
  - Added delay of 1 clock cycle in FLEXIO_I2C_MasterTransferRunStateMachine to ensure that bus would be idle before next transfer if master is nacked.
  - Fixed issue that the restart setup time is less than the time in I2C spec by adding delay of 1 clock cycle before restart signal.

[2.3.0]

- Improvements
  - Used 3 timers instead of 2 to support transfer which is more than 14 bytes in single transfer.
  - Improved FLEXIO_I2C_MasterTransferGetCount so that the API can check whether the transfer is still in progress.
- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.2.0]

- New Features
  - Added timeout mechanism when waiting certain state in transfer API.
  - Added an API for checking bus pin status.
- Bug Fixes
  - Fixed COVERITY issue of useless call in FLEXIO_I2C_MasterTransferRunStateMachine.
  - Fixed MISRA issues
    - Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.
  - Added codes in FLEXIO_I2C_MasterTransferCreateHandle to clear pending NVIC IRQ, disable internal IRQs before enabling NVIC IRQ.
  - Modified code so that during master's nonblocking transfer the start and slave address are sent after interrupts being enabled, in order to avoid potential issue of sending the start and slave address twice.

[2.1.7]

- Bug Fixes
  - Fixed the issue that FLEXIO_I2C_MasterTransferBlocking did not wait for STOP bit sent.
  - Fixed COVERITY issue of useless call in FLEXIO_I2C_MasterTransferRunStateMachine.
  - Fixed the issue that I2C master did not check whether bus was busy before transfer.

[2.1.6]

- Bug Fixes
  - Fixed the issue that I2C Master transfer APIs(blocking/non-blocking) did not support the situation of master transfer with subaddress and transfer data size being zero, which means no data followed the subaddress.

[2.1.5]

- Improvements
  - Unified component full name to FLEXIO I2C Driver.

[2.1.4]

- Bug Fixes
  - The following modifications support FlexIO using multiple instances:
    * Removed FLEXIO_Reset API in module Init APIs.
    * Updated module Deinit APIs to reset the shifter/timer config instead of disabling module/clock.
    * Updated module Enable APIs to only support enable operation.

[2.1.3]

- Improvements
  - Changed the prototype of FLEXIO_I2C_MasterInit to return kStatus_Success if initialized successfully or to return kStatus_InvalidArgument if "(srcClock_Hz / masterConfig->baudRate_Bps) / 2 - 1" exceeds 0xFFU.

[2.1.2]

- Bug Fixes
  - Fixed the FLEXIO I2C issue where the master could not receive data from I2C slave in high baudrate.
  - Fixed the FLEXIO I2C issue where the master could not receive NAK when master sent non-existent addr.
  - Fixed the FLEXIO I2C issue where the master could not get transfer count successfully.
  - Fixed the FLEXIO I2C issue where the master could not receive data successfully when sending data first.
  - Fixed the Dozen mode configuration error in FLEXIO_I2C_MasterInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
  - Fixed the issue that FLEXIO_I2C_MasterTransferBlocking API called FLEXIO_I2C_MasterTransferCreateHandle, which lead to the s_flexioHandle/s_flexioIsr/s_flexioType variable being written. Then, if calling FLEXIO_I2C_MasterTransferBlocking API multiple times, the s_flexioHandle/s_flexioIsr/s_flexioType variable would not be written any more due to it being out of range. This lead to the following situation: NonBlocking transfer APIs could not work due to the fail of register IRQ.

[2.1.1]

- Bug Fixes
  - Implemented the FLEXIO_I2C_MasterTransferBlocking API which is defined in header file but has no implementation in the C file.

[2.1.0]

- New Features
  - Added Transfer prefix in transactional APIs.
  - Added transferSize in handle structure to record the transfer size.

FLEXIO_I2S

[2.2.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 12.4.

[2.2.1]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.2.0]

- New Features
  - Added timeout mechanism when waiting certain state in transfer API.
- Bug Fixes
  - Fixed IAR Pa082 warnings.
  - Fixed violations of the MISRA C-2012 rules 10.4, 14.4, 11.8, 11.9, 10.1, 17.7, 11.6, 10.3, 10.7.

[2.1.6]

- Bug Fixes
  - Added reset flexio before flexio i2s init to make sure flexio status is normal.

[2.1.5]

- Bug Fixes
  - Fixed the issue that I2S driver used hard code for bitwidth setting.

[2.1.4]

- Improvements
  - Unified component's full name to FLEXIO I2S (DMA/EDMA) driver.

[2.1.3]

- Bug Fixes
  - The following modifications support FLEXIO using multiple instances:
    - Removed FLEXIO_Reset API in module Init APIs.
    - Updated module Deinit APIs to reset the shifter/timer config instead of disabling module/clock.
    - Updated module Enable APIs to only support enable operation.

[2.1.2]

- New Features
  - Added configure items for all pin polarity and data valid polarity.
  - Added default configure for pin polarity and data valid polarity.

[2.1.1]

- Bug Fixes
  - Fixed FlexIO I2S RX data read error and eDMA address error.
  - Fixed FlexIO I2S slave timer compare setting error.

[2.1.0]

- New Features
  - Added Transfer prefix in transactional APIs.
  - Added transferSize in handle structure to record the transfer size.

FLEXIO_I2S_EDMA

[2.1.9]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 12.4.

[2.1.8]

- Improvements
  - Applied EDMA ERRATA 51327.

FLEXIO_MCU_LCD

[2.3.0]

- New Features
  - Supported passing an extra user defined parameter to GPIO functions to control the CS/RS/RDWR pin signal.

[2.2.0]

- Improvements
  - Supported platforms which don't have DOZE mode control.

[2.1.0]

- New Features
  - Supported transmit only data without command.

[2.0.8]

- Bug Fixes
  - Fixed bug that FLEXIO_MCULCD_Init return kStatus_Success even with invalid parameter.
  - Fixed glitch on WR, that when initially configure the timer pin as output, or change the pin back to disabled, the pin may be driven low causing glitch on bus. Configure the pin as bidirection output first then perform a subsequent write to change to output or dsiabled to avoid the issue.

[2.0.6]

- Bug Fixes
  - Fixed MISRA 10.4 issues when FLEXIO_MCULCD_DATA_BUS_WIDTH defined as signed value.

[2.0.5]

- Improvements
  - Changed FLEXIO_MCULCD_WriteDataArrayBlocking's data parameter to const type.

[2.0.4]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.0.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 14.4, 17.7.

[2.0.2]

- Improvements
  - Unified component full name to FLEXIO_MCU_LCD (EDMA) driver.

[2.0.1]

- Bug Fixes
  - The following modification to support FlexIO using multiple instances:
    - Removed FLEXIO_Reset API in module Init APIs.
    - Updated module Deinit APIs to reset the shifter/timer configuration instead of disabling module and clock.
    - Updated module Enable APIs to only support enable operation.

[2.0.0]

- Initial version.

FLEXIO_MCU_LCD_EDMA

[2.0.6]

- New Features
  - Supported passing an extra user defined parameter to GPIO functions to control the RDWR pin signal.

[2.0.5]

- New Features
  - Supported transmit only data without command.

[2.0.4]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.0.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.3, 10.4, 10.6, 14.4, 17.7.

[2.0.2]

- Improvements
  - Unified component full name to FLEXIO_MCU_LCD (EDMA) driver.

[2.0.1]

- Bug Fixes
  - The following modification to support FlexIO using multiple instances:
    - Removed FLEXIO_Reset API in module Init APIs.
    - Updated module Deinit APIs to reset the shifter/timer configuration instead of disabling module and clock.
    - Updated module Enable APIs to only support enable operation.

[2.0.0]

- Initial version.

FLEXIO_SPI

[2.4.4]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.4.3]

- Improvements
  - Make SPI_RETRY_TIMES configurable by CONFIG_SPI_RETRY_TIMES.

[2.4.2]

- Bug Fixes
  - Fixed FLEXIO_SPI_MasterTransferBlocking and FLEXIO_SPI_MasterTransferNonBlocking issue in CS continuous mode, the CS might not be continuous.

[2.4.1]

- Bug Fixes
  - Fixed coverity issues

[2.4.0]

- Improvements
  - Supported platforms which don't have DOZE mode control.

[2.3.5]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.3.4]

- Bug Fixes
  - Fixed the txData from void * to const void * in transmit API

[2.3.3]

- Bugfixes
  - Fixed cs-continuous mode.

[2.3.2]

- Improvements
  - Changed FLEXIO_SPI_DUMMYDATA to 0x00.

[2.3.1]

- Bugfixes
  - Fixed IRQ SHIFTBUF overrun issue when one FLEXIO instance used as multiple SPIs.

[2.3.0]

- New Features
  - Supported FLEXIO_SPI slave transfer with continuous master CS signal and CPHA=0.
  - Supported FLEXIO_SPI master transfer with continuous CS signal.
  - Support 32 bit transfer width.
- Bug Fixes
  - Fixed wrong timer compare configuration for dma/edma transfer.
  - Fixed wrong byte order of rx data if transfer width is 16 bit, since the we use shifter buffer bit swapped/byte swapped register to read in received data, so the high byte should be read from the high bits of the register when MSB.

[2.2.1]

- Bug Fixes
  - Fixed bug in FLEXIO_SPI_MasterTransferAbortEDMA that when aborting EDMA transfer EDMA_AbortTransfer should be used rather than EDMA_StopTransfer.

[2.2.0]

- Improvements
  - Added timeout mechanism when waiting certain states in transfer driver.
- Bug Fixes
  - Fixed MISRA 10.4 issues.
  - Added codes in FLEXIO_SPI_MasterTransferCreateHandle and FLEXIO_SPI_SlaveTransferCreateHandle to clear pending NVIC IRQ before enabling NVIC IRQ, to fix issue of pending IRQ interfering the on-going process.

[2.1.3]

- Improvements
  - Unified component full name to FLEXIO SPI(DMA/EDMA) Driver.
- Bug Fixes
  - Fixed MISRA issues
    - Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

[2.1.2]

- Bug Fixes
  - The following modification support FlexIO using multiple instances:
    - Removed FLEXIO_Reset API in module Init APIs.
    - Updated module Deinit APIs to reset the shifter/timer config instead of disabling module/clock.
    - Updated module Enable APIs to only support enable operation.

[2.1.1]

- Bug Fixes
  - Fixed bug where FLEXIO SPI transfer data is in 16 bit per frame mode with eDMA.
  - Fixed bug when FLEXIO SPI works in eDMA and interrupt mode with 16-bit per frame and Lsbfirst.
  - Fixed the Dozen mode configuration error in FLEXIO_SPI_MasterInit/FLEXIO_SPI_SlaveInit API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
- Improvements
  - Added #ifndef/#endif to allow users to change the default TX value at compile time.

[2.1.0]

- New Features
  - Added Transfer prefix in transactional APIs.
  - Added transferSize in handle structure to record the transfer size.
- Bug Fixes
  - Fixed the error register address return for 16-bit data write in FLEXIO_SPI_GetTxDataRegisterAddress.
  - Provided independent IRQHandler/transfer APIs for Master and slave to fix the baudrate limit issue.

FLEXIO_UART

[2.6.5]

- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.6.4]

- Improvements
  - Make UART_RETRY_TIMES configurable by CONFIG_UART_RETRY_TIMES.

[2.6.3]

- Bug Fixes
  - Fixed coverity issues

[2.6.2]

- Bug Fixes
  - Fixed coverity issues

[2.6.1]

- Improvements
  - Improve baudrate calculation method, to support higher frequency FlexIO clock source.

[2.6.0]

- Improvements
  - Supported platforms which don't have DOZE mode control.

[2.5.1]

- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.5.0]

- Improvements
  - Added API FLEXIO_UART_FlushShifters to flush UART fifo.

[2.4.0]

- Improvements
  - Use separate data for TX and RX in flexio_uart_transfer_t.
- Bug Fixes
  - Fixed bug that when ring buffer is used, if some data is received in ring buffer first before calling FLEXIO_UART_TransferReceiveNonBlocking, the received data count returned by FLEXIO_UART_TransferGetReceiveCount is wrong.

[2.3.0]

- Improvements
  - Added check for baud rate's accuracy that returns kStatus_FLEXIO_UART_BaudrateNotSupport when the best achieved baud rate is not within 3% error of configured baud rate.
- Bug Fixes
  - Added codes in FLEXIO_UART_TransferCreateHandle to clear pending NVIC IRQ before enabling NVIC IRQ, to fix issue of pending IRQ interfering the on-going process.

[2.2.0]

- Improvements
  - Added timeout mechanism when waiting for certain states in transfer driver.
- Bug Fixes
  - Fixed MISRA 10.4 issues.

[2.1.6]

- Bug Fixes
  - Fixed IAR Pa082 warnings.
  - Fixed MISRA issues
    - Fixed rules 10.1, 10.3, 10.4, 10.7, 11.6, 11.9, 14.4, 17.7.

[2.1.5]

- Improvements
  - Triggered user callback after all the data in ringbuffer were received in FLEXIO_UART_TransferReceiveNonBlocking.

[2.1.4]

- Improvements
  - Unified component full name to FLEXIO UART(DMA/EDMA) Driver.

[2.1.3]

- Bug Fixes
  - The following modifications support FLEXIO using multiple instances:
    - Removed FLEXIO_Reset API in module Init APIs.
    - Updated module Deinit APIs to reset the shifter/timer configuration instead of disabling module and clock.
    - Updated module Enable APIs to only support enable operation.

[2.1.2]

- Bug Fixes
  - Fixed the transfer count calculation issue in FLEXIO_UART_TransferGetReceiveCount, FLEXIO_UART_TransferGetSendCount, FLEXIO_UART_TransferGetReceiveCountDMA, FLEXIO_UART_TransferGetSendCountDMA, FLEXIO_UART_TransferGetReceiveCountEDMA and FLEXIO_UART_TransferGetSendCountEDMA.
  - Fixed the Dozen mode configuration error in FLEXIO_UART_Init API. For enableInDoze = true, the configuration should be 0; for enableInDoze = false, the configuration should be 1.
  - Added code to report errors if the user sets a too-low-baudrate which FLEXIO cannot reach.
  - Disabled FLEXIO_UART receive interrupt instead of all NVICs when reading data from ring buffer. If ring buffer is used, receive nonblocking will disable all NVIC interrupts to protect the ring buffer. This had negative effects on other IPs using interrupt.

[2.1.1]

- Bug Fixes
  - Changed the API name FLEXIO_UART_StopRingBuffer to FLEXIO_UART_TransferStopRingBuffer to align with the definition in C file.

[2.1.0]

- New Features
  - Added Transfer prefix in transactional APIs.
  - Added txSize/rxSize in handle structure to record the transfer size.
- Bug Fixes
  - Added an error handle to handle the situation that data count is zero or data buffer is NULL.

FLEXIO_UART_EDMA

[2.3.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules.

[2.3.0]

- Refer FLEXIO_UART driver change log to 2.3.0

FREQME

[2.2.0]

- Improvements
  - Add following API to support FREQME which exposes dedicated REF and TAR registers for clock source selection.
    - FREQME_SetReferenceClk
    - FREQME_SetTargetClk
  - Support some devices which apply a different frequency measurement formula (result + 1 instead of result - 2).

[2.1.4]

- Improvements
  - Clarified pulse width measurement mode usage.

[2.1.3]

- Bug Fixes
  - Fixed violations of the CERT INT30-C INT31-C.

[2.1.2]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.1.1]

- Fixed MISRA issues.

[2.1.0]

- Updated register name.

[2.0.0]

- Initial version.

GDET

[2.1.4]

- Add wait into GDET_ReconfigureVoltageMode to allow update conf properly finish

[2.1.3]

- Update to KW43 GDET offering

[2.1.2]

- Update to MCXA577 GDET offering

[2.1.1]

- Update to RT700 GDET offering

[2.1.0]

- Update for multiple instances
- Fix bug in isolation off API
- Add enable and disable APIs

[2.0.1]

- Fix MISRA in GDET_ReconfigureVoltageMode().

[2.0.0]

- Initial version.

GLIKEY

[2.0.1]

- Implement INIT state recovery from the LOCKED state after a reset when the previous index was locked.

[2.0.0]

- Initial version.

GPIO

[2.8.6]

- Bug Fixes
  - Fixed CERT INT31-C violations in GPIO_FIT_REG macro for platforms with 8-bit register width.

[2.8.5]

- Improvements
  - Add AND logic in GPIO_GpioClearInterruptFlags to support clearing multiple interrupt flags simultaneously.

[2.8.4]

- Improvements
  - Make function GPIO_PortGetInterruptFlags and GPIO_PortClearInterruptFlags available for all variants, they are used for all pins flags in one GPIO port.

[2.8.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 5.7.

[2.8.2]

- Bug Fixes
  - Fixed COVERITY issue that GPIO_GetInstance could return clock array overflow values due to GPIO base and clock being out of sync.

[2.8.1]

- Bug Fixes
  - Fixed CERT INT31-C issues.

[2.8.0]

- Improvements
  - Add API GPIO_PortInit/GPIO_PortDeinit to set GPIO clock enable and releasing GPIO reset.

[2.8.0]

- Improvements
  - Add API GPIO_PortInit/GPIO_PortDeinit to set GPIO clock enable and releasing GPIO reset.
  - Remove support for API GPIO_GetPinsDMARequestFlags with GPIO_ISFR_COUNT <= 1.

[2.7.3]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.7.2]

- New Features
  - Support devices without PORT module.

[2.7.1]

- Bug Fixes
  - Fixed MISRA C-2012 rule 10.4 issues in GPIO_GpioGetInterruptChannelFlags() function and GPIO_GpioClearInterruptChannelFlags() function.

[2.7.0]

- New Features
  - Added API to support Interrupt select (IRQS) bitfield.

[2.6.0]

- New Features
  - Added API to get GPIO version information.
  - Added API to control a pin for general purpose input.
  - Added some APIs to control pin in secure and previliege status.

[2.5.3]

- Bug Fixes
  - Correct the feature macro typo: FSL_FEATURE_GPIO_HAS_NO_INDEP_OUTPUT_CONTORL.

[2.5.2]

- Improvements
  - Improved GPIO_PortSet/GPIO_PortClear/GPIO_PortToggle functions to support devices without Set/Clear/Toggle registers.

[2.5.1]

- Bug Fixes
  - Fixed wrong macro definition.
  - Fixed MISRA C-2012 rule issues in the FGPIO_CheckAttributeBytes() function.
  - Defined the new macro to separate the scene when the width of registers is different.
  - Removed some redundant macros.
- New Features
  - Added some APIs to get/clear the interrupt status flag when the port doesn't control pins' interrupt.

[2.4.1]

- Improvements
  - Improved GPIO_CheckAttributeBytes() function to support 8 bits width GACR register.

[2.4.0]

- Improvements
  - API interface added:
    - New APIs were added to configure the GPIO interrupt clear settings.

[2.3.2]

- Bug Fixes
  - Fixed the issue for MISRA-2012 check.
    - Fixed rule 3.1, 10.1, 8.6, 10.6, and 10.3.

[2.3.1]

- Improvements
  - Removed deprecated APIs.

[2.3.0]

- New Features
  - Updated the driver code to adapt the case of interrupt configurations in GPIO module. New APIs were added to configure the GPIO interrupt settings if the module has this feature on it.

[2.2.1]

- Improvements
  - API interface changes:
    - Refined naming of APIs while keeping all original APIs by marking them as deprecated. The original APIs will be removed in next release. The main change is updating APIs with prefix of _PinXXX() and _PortXXX.

[2.1.1]

- Improvements
  - API interface changes:
    * Added an API for the check attribute bytes.

[2.1.0]

- Improvements
  - API interface changes:
    * Added “pins” or “pin” to some APIs’ names.
    * Renamed “_PinConfigure” to “GPIO_PinInit”.

I3C

[2.15.0]

- Improvements
  - Added new timing APIs (I3C_MasterCalcTiming, I3C_MasterSetTiming) and supporting types to provide bus-topology-aware baud rate calculation with ODHPP control and pre-computed timing switching.
- New Features
  - Added I3C_MasterGetI2CPresetTiming to get baud rate settings which meet I2C timing.
- Bug Fixes
  - Fixed Coverity issues.

[2.14.7]

- Bug Fixes
  - Fixed I3C master interrupt transfer not stopping the bus on errors other than Nak/IBIWon.

[2.14.6]

- New Features
  - Added common IRQ handler entry I3C_DriverIRQHandler.

[2.14.5]

- Bug Fixes
  - Fixed CERT INT31-C violation: cast bool field enableSlave to uint via ternary in I3C_Init.

[2.14.4]

- Improvements
  - Fixed MSG violations.

[2.14.3]

- Improvements
  - Fixed Coverity CERT-C violations.
  - Used I3C_RSTS instead of I3C special feature macro.
  - Adapted the driver to support new platform.

[2.14.2]

- Improvements
  - Added timeout for ENTDAA process API.
  - Added build system macro to control the timeout setting.

[2.14.1]

- Improvements
  - Split the function I3C_MasterTransferBlocking to meet the HIS-CCM requirement.

[2.14.0]

- Improvements
  - Added the choice to set fast start header with push-pull speed when all targets addresses have MSB 0 instead of forcing to set it.
  - Deleted duplicated busy check in I3C_MasterStart function.

[2.13.1]

- Bug Fixes
  - Disabled Rx auto-termination in repeated start interrupt event while transfer API doesn't enable it.
  - Waited the completion event after loading all Tx data in Tx FIFO.
- Improvements
  - Conditionally compile interrupt handling code to solve the problem of using this driver on CPU cores that do not support interrupts.

[2.13.0]

- New features
  - Added the hot-join support for I3C bus initialization API.
- Bug Fixes
  - Set read termination with START at the same time in case unknown issue.
  - Set MCTRL[TYPE] as 0 for DDR force exit.
- Improvements
  - Added the API to reset device count assigned by ENTDAA.
  - Provided the method to set global macro I3C_MAX_DEVCNT to determine how many device addresses ENTDAA can allocate at one time.

  - Initialized target management static array based on instance number for the case that multiple instances are used at the same time.

[2.12.0]

- Improvements
  - Added the slow clock parameter for Controller initialization function to calculate accurate timeout.
- Bug Fixes
  - Fixed the issue that BAMATCH field can't be 0. BAMATCH should be 1 for 1MHz slow clock.

[2.11.1]

- Bug Fixes
  - Fixed the issue that interrupt API transmits extra byte when subaddress and data size are null.
  - Fixed the slow clock calculation issue.

[2.11.0]

- New features
  - Added the START/ReSTART SCL delay setting for the Soc which supports this feature.
- Bug Fixes
  - Fixed the issue that ENTDAA process waits Rx pending flag which causes problem when Rx watermark isn't 0. Just check the Rx FIFO count.

[2.10.8]

- Improvements
  - Support more instances.

[2.10.7]

- Improvements
  - Fixed the potential compile warning.

[2.10.6]

- New features
  - Added the I3C private read/write with 0x7E address as start.

[2.10.5]

- New features
  - Added I3C HDR-DDR transfer support.

[2.10.4]

- Improvements
  - Added one more option for master to not set RDTERM when doing I3C Common Command Code transfer.

[2.10.3]

- Improvements
  - Masked the slave IBI/MR/HJ request functions with feature macro.

[2.10.2]

- Bug Fixes
  - Added workaround for errata ERR051617: I3C working with I2C mode creates the unintended Repeated START before actual STOP on some platforms.

[2.10.1]

- Bug Fixes
  - Fixed the issue that DAA function doesn't wait until all Rx data is read out from FIFO after master control done flag is set.
  - Fixed the issue that DAA function could return directly although the disabled interrupts are not enabled back.

[2.10.0]

- New features
  - Added I3C extended IBI data support.

[2.9.0]

- Improvements
  - Added adaptive termination for master blocking transfer. Set termination with start signal when receiving bytes less than 256.

[2.8.2]

- Improvements
  - Fixed the build warning due to armgcc strict check.

[2.8.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 17.7.

[2.8.0]

- Improvements
  - Added API I3C_MasterProcessDAASpecifiedBaudrate for temporary baud rate adjustment when I3C master assigns dynamic address.

[2.7.1]

- Bug Fixes
  - Fixed the issue that I3C slave handle STOP event before finishing data transmission.

[2.7.0]

- Fixed the CCM problem in file fsl_i3c.c.
- Fixed the FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND usage issue in I3C_GetDefaultConfig and I3C_Init.

[2.6.0]

- Fixed the FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND usage issue in fsl_i3c.h.
- Changed some static functions in fsl_i3c.c as non-static and define the functions in fsl_i3c.h to make I3C DMA driver reuse:
  - I3C_GetIBIType
  - I3C_GetIBIAddress
  - I3C_SlaveCheckAndClearError
- Changed the handle pointer parameter in IRQ related funtions to void * type to make it reuse in I3C DMA driver.
- Added new API I3C_SlaveRequestIBIWithSingleData for slave to request single data byte, this API could be used regardless slave is working in non-blocking interrupt or non-blocking dma.
- Added new API I3C_MasterGetDeviceListAfterDAA for master application to get the device information list built up in DAA process.

[2.5.4]

- Improved I3C driver to avoid setting state twice in the SendCommandState of I3C_RunTransferStateMachine.
- Fixed MISRA violation of rule 20.9.
- Fixed the issue that I3C_MasterEmitRequest did not use Type I3C SDR.

[2.5.3]

- Updated driver for new feature FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH and FSL_FEATURE_I3C_HAS_NO_SCONFIG_IDRAND.

[2.5.2]

- Updated driver for new feature FSL_FEATURE_I3C_HAS_NO_MERRWARN_TERM.
- Fixed the issue that call to I3C_MasterTransferBlocking API did not generate STOP signal when NAK status was returned.

[2.5.1]

- Improved the receive terminate size setting for interrupt transfer read, now it's set at beginning of transfer if the receive size is less than 256 bytes.

[2.5.0]

- Added new API I3C_MasterRepeatedStartWithRxSize to send repeated start signal with receive terminate size specified.
- Fixed the status used in I3C_RunTransferStateMachine, changed to use pending interrupts as status to be handled in the state machine.
- Fixed MISRA 2012 violation of rule 10.3, 10.7.

[2.4.0]

- Bug Fixes
  - Fixed kI3C_SlaveMatchedFlag interrupt is not properly handled in I3C_SlaveTransferHandleIRQ when it comes together with interrupt kI3C_SlaveBusStartFlag.
  - Fixed the inaccurate I2C baudrate calculation in I3C_MasterSetBaudRate.
  - Added new API I3C_MasterGetIBIRules to get registered IBI rules.
  - Added new variable isReadTerm in struct _i3c_master_handle for transfer state routine to check if MCTRL.RDTERM is configured for read transfer.
  - Changed to emit Auto IBI in transfer state routine for slave start flag assertion.
  - Fixed the slave maxWriteLength and maxReadLength does not be configured into SMAXLIMITS register issue.
  - Fixed incorrect state for IBI in I3C master interrupt transfer IRQ handle routine.
  - Added isHotJoin in i3c_slave_config_t to request hot-join event during slave init.

[2.3.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 17.7.
  - Fixed incorrect HotJoin event index in I3C_GetIBIType.

[2.3.1]

- Bug Fixes
  - Fixed the issue that call of I3C_MasterTransferBlocking/I3C_MasterTransferNonBlocking fails for the case which receive length 1 byte of data.
  - Fixed the issue that STOP signal is not sent when NAK status is detected during execution of I3C_MasterTransferBlocking function.

[2.3.0]

- Improvements
  - Added I3C common driver APIs to initialize I3C with both master and slave configuration.

  - Updated I3C master transfer callback to function set structure to include callback invoke for IBI event and slave2master event.
  - Updated I3C master non-blocking transfer model and always enable the interrupts to be able to re-act to the slave start event and handle slave IBI.

[2.2.0]

- Bug Fixes
  - Fixed the issue that I3C transfer size limit to 255 bytes.

[2.1.2]

- Bug Fixes
  - Reset default hkeep value to kI3C_MasterHighKeeperNone in I3C_MasterGetDefaultConfig

[2.1.1]

- Bug Fixes
  - Fixed incorrect FIFO reset operation in I3C Master Transfer APIs.
  - Fixed i3c slave IRQ handler issue, slave transmit could be underrun because tx FIFO is not filled in time right after start flag detected.

[2.1.0]

- Added definitions and APIs for I3C slave functionality, updated previous I3C APIs to support I3C functionality.

[2.0.0]

- Initial version.

I3C_EDMA

[2.2.13]

- Changed
  - Use linked TCD for the slave TX end byte instead of writing the last byte in the EDMA callback.

[2.2.12]

- Bug Fixes
  - Fixed the issue that EDMA transfer configuration use wrong parameter.

[2.2.11]

- Improvements
  - Fixed Coverity CERT-C violations.

[2.2.10]

- Bug Fixes
  - Fixed the issue that slave start event is cleared when it has not been handled.
- Added
  - Supported I3C HDR-DDR transfer with EDMA.
- Changed
  - Used linked EDMA to transfer all I3C subaddress and data without handling of intermediate states, simplifying code logic.
  - Prepare DMA before I3C START to ensure there's no time delay between START and transmitting data.
  - Added the MCTRLDONE flag check after START and STOP request to ensure all states are handled properly.

[2.2.9]

- Bug Fixes
  - Fixed MISRA issue rule 11.3.
  - Added the master control done flag waiting code after STOP in case the bus is not idle when transfer function finishes.

[2.2.8]

- Improvements
  - Removed I3C IRQ handler calling in the EDMA callback. Previously driver doesn't use the END byte which can trigger the STOP interrupt for controller sending and receiving, now let I3C event handler deal with all I3C events.
- Bug Fixes
  - Fixed the bug that the END type Tx register is not used when command length or data length is one byte.

[2.2.7]

- Bug Fixes
  - Fixed MISRA issue rule 11.6.

[2.2.6]

- New features
  - Added the I3C private read/write with 0x7E address as start.

[2.2.5]

- Improvements
  - Added the workaround for RT1180 I3C EDMA issue ERR052086.

[2.2.4]

- Bug Fixes
  - Fixed the issue that I3C master sends the last byte data without using the END type register.

[2.2.3]

- Bug Fixes
  - Fixed issue that slave polulates the last byte when Tx FIFO may be full.

[2.2.2]

- Bug Fixes
  - Fixed I3C MISRA issue rule 10.4, 11.3.

[2.2.1]

- Bug Fixes
  - Fixed the issue that I3C slave send the last byte data without using the END type register.
- Improvements
  - There's no need to reserve two bytes FIFO for DMA transfer which is for IP issue workaround.

[2.2.0]

- Improvements
  - Deleted legacy IBI data request code.

[2.1.0]

- Bug Fixes
  - Fixed MISRA issue rule 8.4, 8.6, 11.8.

[2.0.1]

- Bug Fixes
  - Fixed MISRA issue rule 9.1.

[2.0.0]

- Initial version.

INPUTMUX

[2.0.11]

- Bug Fixes
  - Fixed Coverity UNUSED_VALUE finding in INPUTMUX_EnableSignal().

[2.0.10]

- Bug Fixes
  - Fixed CERT-C violations.

[2.0.9]

- Improvements
  - Use INPUTMUX_CLOCKS to initialize the inputmux module clock to adapt to multiple inputmux instances.
  - Modify the API base type from INPUTMUX_Type to void.

[2.0.8]

- Improvements
  - Updated a feature macro usage for function INPUTMUX_EnableSignal.

[2.0.7]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.0.6]

- Bug Fixes
  - Fixed the documentation wrong in API INPUTMUX_AttachSignal.

[2.0.5]

- Bug Fixes
  - Fixed build error because some devices has no sct.

[2.0.4]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rule 10.4, 12.2 in INPUTMUX_EnableSignal() function.

[2.0.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.4, 10.7, 12.2.

[2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.4, 12.2.

[2.0.1]

- Support channel mux setting in INPUTMUX_EnableSignal().

[2.0.0]

- Initial version.

IOPCTL

[2.0.3]

- Improvement
  - Added IOPCTL_FULLDRIVE_EN for full driver enablement.
  - Fixed MISRA C-2012 10.3 issues.

[2.0.2]

- Bug fixes
  - Fix MISRA issues.

[2.0.1]

- Bug fixes
  - Remove unsupported IOPCTL_FULLDRIVE_EN(bit8) configuration.

[2.0.0]

- Initial version.

IRTC

[2.3.4]

- Bug Fixes
  - Fixed CERT INT31-C violations to ensure safe integer conversions.

[2.3.3]

- Bug Fixes
  - Fix CERT INT31-C issue.

[2.3.2]

- Bug Fixes
  - Fixed API IRTC_GetDatetime read YEARMON, DAYS, HOURMIN, SECONDS registers issue.

[2.3.1]

- Bug Fixes
  - Fixed MISRA C-2012 issue 10.4.

[2.3.0]

- New Feature
  - Supported platforms with multiple IRTC instances.

[2.2.4]

- Bug Fixes
  - Fixed MISRA C-2012 issue 10.1, 10.3, 10.4, 10.7, 12.2.

[2.2.3]

- Bug Fixes
  - Updated undefined macro names by available ones.

[2.2.2]

- Bug Fixes
  - Fixed MISRA C-2012 issue 10.3.

[2.2.1]

- Bug Fixes
  - Fixed MISRA issues.

[2.2.0]

- New Feature
  - Add new APIs for CLK_SEL and CLKO to select RTC clock and enable/disable output to peripherals.
  - Supported platforms without tamper feature.

[2.1.3]

- Bug Fixes
  - Fixed MISRA C-2012 issue 10.1 and 10.4.

[2.1.2]

- Bug Fixes
  - Fixed kIRTC_TamperFlag flag can't be cleared issue.

[2.1.1]

- Bug Fixes
  - MISRA C-2012 issue check.
    - Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4.

[2.1.0]

- Bug Fixes
  - Fixed incorrect leap year check in IRTC_CheckDatetimeFormat.
- New Feature
  - Added new APIs for new feature FSL_FEATURE_RTC_HAS_SUBSYSTEM.
  - Added new APIs for TAMPER, TAMPER QUEUE status get and clear.
  - Added new API to enable/disable 32 kHz RTC OSC clock during RTC register write.
  - Updated IRTC_SetTamperParams to support new feature FSL_FEATURE_RTC_HAS_FILTER23_CFG
  - Updated irtc_config_t to exclude member wakeupSelect for new feature FSL_FEATURE_RTC_HAS_NO_CTRL2_WAKEUP_MODE.

[2.0.2]

- Bug Fixes
  - MISRA C-2012 issue check.
    - Fixed rules, containing: rule-10.1, rule-10.3, rule-10.4, rule-10.6, rule-10.8, rule-11.9, rule-12.2, rule-15.5, rule-16.4, rule-17.7.

[2.0.1]

- Bug Fixes
  - Fixed the issue of hard code in IRTC_Init.

[2.0.0]

- Initial version.

ITRC

[2.4.2]

- Update to KW43 ITRC offering

[2.4.1]

- Update to MCXA577 ITRC offering

[2.4.0]

- Rework the input signal definition for better flexibility

[2.3.0]

- Update names of kITRC_SwEvent1/2 to kITRC_SwEvent0/1 to align with RM

[2.2.0]

- Update driver to new version and input events

[2.1.0]

- Make SYSCON glitch platform dependent.

[2.0.0]

- Initial version.

JPEGDEC

[2.0.2]

- Other Changes
  - Update driver to support more resolution due to the IP bug fix.

[2.0.1]

- Bug Fixes
  - Fixed CERT-C violations.

[2.0.0]

- Initial version.

LCDIF

[2.3.1]

- Bug Fixes
  - Fixed CERT-C violations.
  - Fixed wrong frame buffer configuration for overlay layers.

[2.3.0]

- New Features
  - Supported layer decompress mode for DC8000.

[2.2.0]

- New Features
  - Supported new layers and configurations for DC8000.
  - Added new APIs and configurations to support DBI interface.
- Bug Fixes
  - Update align calculation method, the old one can only be used when the align bytes' low bits are all zeros.

[2.1.2]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.1.1]

- Improvements
  - Added memory address conversion to support buffers which could only be accessed using alias address by non-core masters.
- Bug Fixes
  - Fix MISRA-C 2012 issues.

[2.1.0]

- Bug Fixes
  - Corrected the frame buffer pixel format name.

[2.0.0]

- Initial version.

LPADC

[2.11.0]

- New Features
  - Added the LPADC_USE_FIXED_POINT_GAIN_CALCULATION configuration macro. When set to 1, LPADC_FinishAutoCalibration() calculates the gain conversion result with integer-only arithmetic instead of the float/double software library, saving several kB of flash on code size sensitive devices. The fixed point result keeps the gain fraction in Q17.14, so the two least significant bits of the gain conversion result are always zero.

[2.10.3]

- Bug Fixes
  - Fixed MISRA C-2012 essential-type-model violations (Rules 10.1, 10.3, 10.4 and 10.8) in LPADC_FinishAutoCalibration and LPADC_GetOffsetValue.

[2.10.2]

- Improvements
  - Gate the trigger (TCTRL) enums and the trigger-id assert on FSL_FEATURE_ADC_TCTRL_COUNT for every trigger index, so devices with fewer triggers no longer expose unavailable trigger registers.

[2.10.1]

- Improvements
  - Reworked the gain-calibration ratio in LPADC_FinishAutoCalibration to use single-precision (float) arithmetic instead of double-precision, removing the dependency on the libgcc double soft-float routines (__floatsidf, __aeabi_dsub, __aeabi_ddiv, __aeabi_d2f) that were linked into every application using the LPADC driver, reducing ROM footprint.

[2.10.0]

- Improvements
  - Use #if !defined(ADC_TCTRL_COUNT) || (ADC_TCTRL_COUNT > nU) to wrap some configurations because these configurations are invalid/illegal on some SoCs.

[2.9.5]

- Improvements
  - Fix doxygen issue, grouping command should be balanced.

[2.9.4]

- Improvements
  - Update LPADC_GetDefaultConfig, change default conversionAverageMode value to: kLPADC_ConversionAverage128 for 3 bit width. kLPADC_ConversionAverage1024 for 4 bit width.

[2.9.3]

- Improvements
  - Add timeout for while loop code.

[2.9.2]

- Improvements
  - Fixed CERT-C issues.

[2.9.1]

- Bug Fixes
  - Fixed incorrect channel B FIFO selection logic.

[2.9.0]

- Bug Fixes
  - Add code to handle the case where GCC[GAIN_CAL] is a signed number.
  - Split LPADC_FinishAutoCalibration function into two functions.
  - Improved LPADC driver.

[2.8.4]

- Bug Fixes
  - Remove function 'LPADC_SetOffsetValue' assert statement, this statement may cause runtime errors in existing code.

[2.8.3]

- Bug Fixes
  - Fixed SDK lpadc driver examples compile issue, move condition 'commandId < ADC_CV_COUNT' to a more appropriate location.

[2.8.2]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 18.1, 10.3, 10.1 and 10.4.

[2.8.1]

- Bug Fixes
  - Fixed LPADC sample mode enum name mistake.

[2.8.0]

- Improvements
  - Release peripheral from reset if necessary in init function.
- Bug Fixes
  - Fixed function LPADC_GetConvResult() issue.
  - Fixed function LPADC_SetConvCommandConfig() bugs.

[2.7.2]

- Improvements
  - Use feature macros instead of header file macros.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 10.1, 10.3, 10.4 and 14.3.

[2.7.1]

- Improvements
  - Corrected descriptions of several functions.
  - Improved function LPADC_GetOffsetValue and LPADC_SetOffsetValue.
  - Revert changes of feature macros for lpadc.
  - Use feature macros instead of header file macros.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rule 10.8.
  - Fixed the violations of MISRA C-2012 rule 10.1, 10.3, 10.4 and 14.3.

[2.7.0]

- Improvements
  - Added supports of CFG2 register.
  - Removed some useless macros.

[2.6.2]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules.
  - Fixed LPADC driver code compile error issue.

[2.6.1]

- Improvements
  - Updated the use of macros in the driver code.

[2.6.0]

- Improvements
  - Added the API LPADC_SetOffset12BitValue() to configure 12bit ADC conversion offset trim value manually.
  - Added the API LPADC_SetOffset16BitValue() to configure 16bit ADC conversion offset trim value manually.
  - Added API to set offset calibration mode.
  - Added configuration of alternate channel.
  - Updated auto calibration API and added calibration value conversion API.
- New feature
  - Added API LPADC_EnableHardwareTriggerCommandSelection() to enable trigger commands controlled by ADC_ETC.
  - Updated LPADC_DoAutoCalibration() to allow doing something else before the ADC initialization to be totally complete. Enhance initialization duration time of the ADC.
  - Added two new APIs to get/set calibration value.

[2.5.2]

- Improvements
  - Added while loop, LPADC_GetConvResult() will return only when the FIFO will not be empty.

[2.5.1]

- Bug Fixes
  - Fixed some typos in Lpadc driver comments.

[2.5.0]

- Improvements
  - Added missing items to enable trigger interrupts.

[2.4.0]

- New features
  - Added APIs to get/clear trigger status flags.

[2.3.0]

- Improvements
  - Removed LPADC_MeasureTemperature() function for the LPADC supports different temperature sensor calculation equations.

[2.2.1]

- Improvements
  - Optimized LPADC_MeasureTemperature() function to support the specific series with flash solidified calibration value.
  - Clean doxygen warnings.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3, rule 10.8 and rule 17.7.

[2.2.0]

- New Feature
  - Added API LPADC_MeasureTemperature() to get correct temperature from the internal sensor.
- Improvements
  - Separated lpadc_conversion_resolution_mode_t with related feature macro.
- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules:
    - Rule 10.3, 10.4, 10.6, 10.7 and 17.7.

[2.1.1]

- Improvements
  - Updated the gain calibration formula.
  - Used feature to segregate the new item kLPADC_TriggerPriorityPreemptSubsequently.

[2.1.0]

- New Features
  - Added the API LPADC_SetOffsetValue() to support configure offset trim value manually.
  - Added the API LPADC_DoOffsetCalibration() to do offset calibration independently.
- Improvements
  - Improved the usage of macros and removed invalid macros.

[2.0.2]

- Improvements
  - Added support for platforms with 2 FIFOs and different calibration measures.

[2.0.1]

- Bug Fixes
  - Ensured the API LPADC_SetConvCommandConfig configure related registers correctly.

[2.0.0]

- Initial version.

LPFLEXCOMM

[2.2.1]

- Bug Fix
  - Fixed the bug that function LPFLEXCOMM_GetBaseAddr() type was uint32_t,but return NULL((void*)0).

[2.2.0]

- New Features
  - Added new API to get LPFLEXCOMM base address.

[2.1.2]

- Bug Fix
  - Fixed the bug that when the same instance is initialized multiple times, all previous states will be cleared and only the last state will be retained.

[2.1.1]

- New Features
  - Supported new platform that has more lpflexcomm instance.

[2.1.0]

- Improvements
  - Function LP_FLEXCOMM_SetPeriph changed from public to private,function LP_FLEXCOMM_Init no longer judges the periph.

[2.0.0]

- Initial version.

LPI2C

[2.2.11]

- Bug Fixes
  - LPI2C_MasterTransferBlocking - added waiting while TX FIFO is not empty.
  - Added a check for an error from the previous transfer in the transfer API (when kLPI2C_TransferNoStopFlag was used).
  - Updated LPI2C_MasterReceive() to return kStatus_InvalidArgument when rxSize is 0.
  - Updated transfer API functions to return kStatus_InvalidArgument when transfer->dataSize is 0 and transfer->direction is kLPI2C_Read.

[2.2.10]

- Bug Fixes
  - Fixed LPI2C_MasterStop() to send I2C STOP in case of error.

[2.2.9]

- Bug Fixes
  - Set handle->chunkSize to 0 in LPI2C_MasterTransferNonBlocking().

[2.2.8]

- Bug Fixes
  - Removed write to STAR register in LPI2C_SlaveTransferHandleIRQ handler in case of kLPI2C_SlaveRepeatedStartDetectFlag or kLPI2C_SlaveStopDetectFlag.
  - Removed configuration of FILTSDA from LPI2C_MasterSetBaudRate().
  - Set minimal possible value for FILTSDA and FILTSCL to 0.
  - Updated configuration of glitch filters in LPI2C_SlaveInit().
  - Set minimal possible value for SCFGR2[CLKHOLD] to 1.
  - Added error check to while loop in LPI2C_RunTransferStateMachine() for case kSendCommandState.

- Improvements
  - Removed receive data size limitation in LPI2C_MasterTransferBlocking() and LPI2C_MasterTransferNonBlocking().

[2.2.7]

- Bug Fixes
  - Limited value of filtSda in LPI2C_MasterSetBaudRate().
  - Fixed configuration of MCFGR2 (FILTSDA) in LPI2C_MasterInit() - the change of FILTSDA in LPI2C_MasterSetBaudRate() could be overwritten.
  - Updated LPI2C_MasterStop() and LPI2C_MasterTransferBlocking() to send I2C STOP also in case of error.
  - Removed unused flag kLPI2C_TransferRepeatedStartFlag.

[2.2.6]

- Bug Fixes
  - Fixed coverity issues.

[2.2.5]

- Improvements
  - Added assert with target feature check in LPI2C_SlaveInit().
  - Added feature check to LPI2C_CommonIRQHandler().

[2.2.4]

- Bug Fixes
  - Fixed LPI2C_MasterTransferBlocking() - the return value was sometime affected by call of LPI2C_MasterStop().

[2.2.3]

- Bug Fixes
  - Fixed an issue that LP_FLEXCOMM_Deinit() is called incorrectly.

[2.2.2]

- Improvements
  - Fixed doxygen warning in LPI2C_SlaveTransferHandleIRQ.

[2.2.1]

- Bug Fixes
  - Added bus stop incase of bus stall in LPI2C_MasterTransferBlocking.

[2.2.0]

- Improvements
  - Support the normal LPI2C in LPFLEXCOMM driver.

[2.1.1]

- Improvements
  - Optimize slave ISR.When replying to ack/nack,first judge whether the user performs the reply in the APP.

[2.1.0]

- New Features
  - Added new function LPI2C_SlaveEnableAckStall to enable or disable ACKSTALL.

[2.0.1]

- Improvements
  - Supported to initialize the flexcomm layer outside the peripheral driver initialization function.

[2.0.0]

- Initial version.

LPI2C_EDMA

[2.0.1]

- Improvements
  - Add EDMA ext API to accommodate more types of EDMA.

[2.0.0]

- Initial version.

LPSPI

[2.2.12]

- Bug Fixes
  - Fixed timing parameter setting sequence issue in $\mathrm{LPSPI_MasterInit}$, the timing parameter should only be configured when LPSPI master is disabled.
- Improvements
  - Improved performance of LPSPI_MasterTransferBlocking().

[2.2.11]

- Bug Fixes
  - Fixed calculation of SCK-to-SCK delay.

[2.2.10]

- Bug Fixes
  - Clear WIDTH bits from the TCR register before writing a new value in LPSPI_MasterTransferBlocking().

[2.2.9]

- Bug Fixes
  - Fixed coverity issues.

[2.2.8]

- Bug Fixes
  - Fixed coverity issues.

[2.2.7]

- Bug Fixes
  - Fixed reading of TCR register
  - Workaround for errata ERR050606

[2.2.6]

- Bug Fixes
  - Fixed an issue that LP_FLEXCOMM_Deinit() is called incorrectly.

[2.2.5]

- Bug Fixes
  - Fixed the txData from void * to const void * in transmit API.

[2.2.4]

- Improvements
  - Fixed doxygen warning in LPSPI_SlaveTransferHandleIRQ.

[2.2.3]

- Bug Fixes
  - Disabled lpspi before LPSPI_MasterSetBaudRate incase of LPSPI opened.

[2.2.2]

- Bug Fixes
  - Fixed 3-wire txmask of handle vaule reentrant issue.

[2.2.1]

- Bug Fixes
  - Workaround for errata ERR051588 by clearing FIFO after transmit underrun occurs.

[2.2.0]

- Feature
  - Added the new feature of multi-IO SPI .

[2.1.1]

- Fixed LPSPI_MasterGetDefaultConfig incorrect default inter-transfer delay calculation.

[2.0.0]

- Initial version.

LPSPI_EDMA

[2.1.4]

- Improvements
  - Reduced DMA requests - transfer up to 4 FIFO words per DMA request

[2.1.3]

- Improvements
  - Increased transmit FIFO watermark to ensure whole transmit FIFO will be used during data transfer.

[2.1.2]

- Bug Fixes
  - Fixed reading of TCR register
  - Workaround for errata ERR050606

[2.1.1]

- Improvements
  - Add EDMA ext API to accommodate more types of EDMA.

[2.1.0]

- Improvements
  - Separated LPSPI_MasterTransferEDMA functions to LPSPI_MasterTransferPrepareEDMA and LPSPI_MasterTransferEDMALite to optimize the process of transfer.

[2.0.0]

- Initial version.

LPUART

[2.3.4]

- Bug Fixes
  - Fixed coverity issues.

[2.3.3]

- Bug Fixes
  - Fixed coverity issues.

[2.3.2]

- Bug Fix
  - Fixed the bug that LPUART_TransferEnable16Bit controled by wrong feature macro.

[2.3.1]

- Bug Fix
  - Fixed MISRA C-2012 violations.

[2.3.0]

- Improvements
  - Added support of DATA register for 9bit or 10bit data transmit in write and read API. Such as: LPUART_WriteBlocking16bit, LPUART_ReadBlocking16bit, LPUART_TransferEnable16Bit LPUART_WriteNonBlocking16bit, LPUART_ReadNonBlocking16bit.

[2.2.4]

- Bug Fix
  - Fixed the bug that baud rate calculation overflow when srcClock_Hz is 528MHz.

[2.2.3]

- Improvements
  - Added atomic in LPUART_EnableInterrupts and LPUART_DisableInterrupts.

[2.2.2]

- Improvements
  - Added comment on txExtendedTimeoutValue of lpuart_timeout_config_t.

[2.2.1]

- Bug Fix
  - Fixed the bug that the OSR calculation error when lupart init and lpuart set baud rate.

[2.2.0]

- Improvements
  - Rename some enumeration variables to be consistent with lpuart driver.

[2.1.1]

- Improvements
  - Supported to initialize the flexcomm layer outside the peripheral driver initialization function.

[2.1.0]

- New Features
  - Supported new platform that does not have the feature of MODEM Control,MODEM Status,Receiver Extended Idle,Transmitter Extended Idle, Half Duplex Control,Timeout Control,Timeout Status and Timeout N.

[2.0.0]

- Initial version.

LPUART_EDMA

[2.1.2]

- Bug Fixes
  - Fixed race condition in LPUART_ReceiveEDMA() for transfers whose size is not a multiple of 4 bytes.

[2.1.1]

- Improvements
  - Reduced DMA requests - transfer up to 4 FIFO words per DMA request

[2.1.0]

- Improvements
  - Added LPUART_TransferCreateHandleEDMAExt - allows configuration of when the TX transfer callback is triggered.

[2.0.0]

- Initial version.

MIPI_DSI

[2.3.1]

- New Features
  - Added common IRQ handler entry MIPI_DSI_CommonDriverIRQHandler.

[2.3.0]

- Bug Fixes
  - Fixed typo in member of dsi_transfer_t structure. The sendDscCmd and dscCmd shall be sendDcsCmd and dcsCmd.

[2.2.2]

- Bug Fixes
  - Fixed CERT-C violations.

[2.2.1]

- Bug Fixes
  - Fixed issue that VACTIVE setting shall equal to the number of active lines (height), no need to minus 1.
- Improvements
  - Update DSI_Deinit to reset peripheral.
  - Update DSI_DeinitDphy to power down DPHY using DPHY_PD_REG before powering down PLL.

[2.2.0]

- New Features
  - Added APIs to configure DBI FIFO and payload.
  - Supported new controls and configurations of DBI pixel format, PHY ready and ULPS for RT700.
  - Updated the DPI setting to use float for coefficient value for more accurate calculation.

[2.1.6]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.1.5]

- Other Changes
  - Changed to use new register naming.
  - Added workaround for Errata ERR011439. Avoid DCS long packet command writes with zero-length data payload in low-power mode, because the checksum is incorrect in this case.

[2.1.4]

- Bug Fixes
  - Fixed the MISRA issues.

[2.1.3]

- Bug Fixes
  - Fixed the DPI horizontal timing setting issue.

[2.1.2]

- Improvements
  - Supported long package read.
- Bug Fixes
  - Fixed the bug that runs to hardfault when sending long packet with 4-byte unaligned address.

[2.1.1]

- Improvements
  - Some SOC compatibility improvement.

[2.1.0]

- Improvements
  - Improved for the platforms which does not support ULPS.

[2.0.6]

- Bug Fixes
  - Fixed the timing issue that non-continuous HS clock mode does not work.

[2.0.5]

- Bug Fixes
  - Fixed kDSI_InterruptGroup1BtaTo and kDSI_InterruptGroup1HtxTo definition error.
- Improvements
  - Changed to override MIPI_DriverIRQHandler instead of MIPI_IRQHandler.

[2.0.4]

- Bug Fixes
  - Fixed MISRA C-2012 issues: 10.1, 10.3, 10.4, 10.4, 10.6, 10.7, 10.8, 11.3, 11.8, 12.2, 14.4, 16.4, 17.7.

[2.0.3]

- Improvement
  - Updated for combo phy header file.

[2.0.2]

- New Features
  - Supported sending separate DSI command from TX data array.
- Bug Fixes
  - Disabled all interrupts in DSI_Init.

[2.0.1]

- Improvements
  - Updated to support the DPHY which does not have internal DPHY PLL.

[2.0.0]

- Initial version.

MMU

[2.0.1]

- Bug Fixes
  - Add assertions to prevent unsigned integer overflow in narrowing conversions from uint32_t to uint16_t
  - Add safe type conversion with temporary variables for address-to-LUT operations
  - Add bounds checking for unit count calculations in region mapping operations
  - Follow INT31-C CERT coding standard for safe narrowing conversions

[2.0.0]

- Initial version.

MRT

[2.0.5]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.0.4]

- Improvements
  - Don't reset MRT when there is not system level MRT reset functions.

[2.0.3]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1 and 10.4.
  - Fixed the wrong count value assertion in MRT_StartTimer API.

[2.0.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4.

[2.0.1]

- Added control macro to enable/disable the RESET and CLOCK code in current driver.

[2.0.0]

- Initial version.

MU

[2.8.5]

- Improvements
  - Guard CIER0 core-interrupt register access with FSL_FEATURE_MU_NO_CIER0_REG for devices whose MU instance has no CIER0 register (e.g. MCXE32B MU2).

[2.8.4]

- Improvements
  - Minor update to allow use in DSC projects.

[2.8.3]

- Improvements
  - Fix build failure when driver clock control disabled.
  - Make MU_GetInstance() static function

[2.8.2]

- Improvements
  - Fix MISRA C-2012 Rule 20.8 violations.

[2.8.1]

- Bug Fixes
  - Avoid incorrect MU_BUSY_POLL_COUNT macro use.

[2.8.0]

- New Features
  - Added MU1_BUSY_POLL_COUNT parameter to prevent infinite polling loops in MU operations.
  - Added timeout mechanism to all polling loops in MU driver code.
- Improvements
  - Updated function signatures to return status codes for better error handling:
    - Changed MU_ResetBothSides to return status_t instead of void
    - Updated MU_SendMsg to return status_t for timeout indication
    - Added new function MU_ReceiveMsgTimeout() to include timeout mechanism.
  - Enhanced documentation across all functions to clarify timeout behavior and return values.

[2.7.0]

- New Features
  - Added API MU_GetRxStatusFlags.

[2.6.0]

- New Features
  - Added API MU_GetInterruptsPending.

[2.5.1]

- Bug Fixes
  - Fixed the bug that MU_TriggerGeneralPurposeInterrupts and MU_TriggerInterrupts may trigger previous triggered general purpose interrupts again by mistake.

[2.5.0]

- New Features
  - Supported more than 4 general purpose interrupts.
  - Added seperate APIs for general purpose interrupts.

[2.4.0]

- Improvements
  - Supported the case that some features only avaiable with specific instances. These features include Hardware Reset, Boot Peer Core, Hold Reset. When using the features with instances which don't support them, driver will report error.

[2.3.3]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.3.2]

- Improvements
  - Supported platforms which don't have CCR0[RSTH], CCR0[CLKE], CCR0[HR], CCR0[HRM].

[2.3.1]

- Bug Fixes
  - Fixed build error for platforms which have CCR0[RSTH], but no CCR0[NMI].

[2.3.0]

- New features
  - Added support for i.MX RT7xx.

[2.2.1]

- Bug Fixes
  - Fixed issue that MU_GetInstance() is defined but never used.

[2.2.0]

- New features
  - Added support for i.MX RT118x.
- Bug Fixes
  - Fixed general purpose interrupt bug.
- Other Changes
  - Change _mu_interrupt_trigger item value.

[2.1.2]

- Bug Fixes
  - Fixed bug that general purpose interrupt can't be configured.

[2.1.1]

- Bug Fixes
  - Fixed MISRA C-2012 issues.

[2.1.0]

- Improvements
  - Added new enum mu_msg_reg_index_t.

[2.0.0]

- Initial version.

OSTIMER

[2.3.1]

- New Features
  - Added common IRQ handler entry OSTIMER_DriverIRQHandler.

[2.3.0]

- New Features
  - Added OSTIMER_SetMatchValueSafe() with post-write synchronization and missed-event detection. After writing the match registers, it waits for OSEVENT_CTRL[MATCH_WR_RDY] to clear, then checks whether the match moment has already passed:
    - Returns kStatus_Success and enables the interrupt if the match is still in the future, or if the hardware interrupt flag is already set (ISR fires immediately on enable).
    - Returns kStatus_Fail and leaves the interrupt **disabled** if the match was missed without the hardware capturing it; the caller must handle the missed event.
  - **API selection guide —** OSTIMER_SetMatchValue **vs** OSTIMER_SetMatchValueSafe**:**
    - Use OSTIMER_SetMatchValue() when the delta between now and count is reliably larger than ~7 OSTimer ticks (e.g. > 200 µs at 32 kHz). It omits the post-write MATCH_WR_RDY poll and missed-event check, so it returns faster. It's suitable for **most use cases**.
    - Use OSTIMER_SetMatchValueSafe() when the match delta is small or unpredictable. The extra synchronization cost (~7 OSTimer ticks) is the trade-off for a guaranteed success/fail result.
- Bug Fixes
  - Fixed OSTIMER_GetMatchValue() missing binary-encoded counter guard; it now returns the raw binary value on devices with binary-encoded registers.
- Improvements
  - Removed useless tmp variable in OSTIMER_SetMatchRawValue().
  - Clarified return value type for counter get APIs.

[2.2.6]

- Improvements
  - Drop the check of MATCH_WR_RDY and ostimer counter value in OSTIMER_SetMatchRawValue. In most applications, they are useless and may bring at least 7 OSTimer ticks latency, which is unacceptable when OSTimer is working under slow peripheral clock.
  - Optimize software gray code to binary conversion: replaced loop-based implementation with branchless bitwise operations.

[2.2.5]

- Improvements
  - Support binary encoded ostimer.

[2.2.4]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.2.3]

- Improvements
  - Disable and clear pending interrupts before disabling the OSTIMER clock to avoid interrupts being executed when the clock is already disabled.

[2.2.2]

- Improvements
  - Support devices with different OSTIMER instance name.

[2.2.1]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.2.0]

- Improvements
  - Move the PMC operation out of the OSTIMER driver to board specific files.
  - Added low level APIs to control OSTIMER MATCH and interrupt.

[2.1.2]

- Bug Fixes
  - Fixed MISRA-2012 rule 10.8.

[2.1.1]

- Bug Fixes
  - removes the suffix ‘n’ for some register names and bit fields’ names
- Improvements
  - Added HW CODE GRAY feature supported by CODE GRAY in SYSCTRL register group.

[2.1.0]

- Bug Fixes
  - Added a workaround to fix the issue that no interrupt was reported when user set smaller period.
  - Fixed violation of MISRA C-2012 rule 10.3 and 11.9.
- Improvements
  - Added return value for the two APIs to set match value.
    - OSTIMER_SetMatchRawValue
    - OSTIMER_SetMatchValue

[2.0.3]

- Bug Fixes
  - Fixed violation of MISRA C-2012 rule 10.3, 14.4, 17.7.

[2.0.2]

- Improvements
  - Added support for OSTIMER0

[2.0.1]

- Improvements
  - Removed the software reset function out of the initialization API.
  - Enabled interrupt directly instead of enabling deep sleep interrupt. Users need to enable the deep sleep interrupt in application code if needed.

[2.0.0]

- Initial version.

PDM

[2.9.6]

- Improvements
  - Added instance-aware dispatch for PDM EVENT, PDM ERROR, HWVAD EVENT, and HWVAD ERROR IRQ handlers while preserving existing wrapper compatibility.

[2.9.5]

- Improvements
  - Updated PDM_ReadFifo shifting logic for variable dataWidth (4/3/2/1 -> shift 0/8/16/24).

[2.9.4]

- Improvements
  - support multiple IRQ handlers

[2.9.3]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

[2.9.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

[2.9.1]

- Bug Fixes
  - Fixed the issue that the driver still enters the interrupt after disabling clock.

[2.9.0]

- Improvements
- Added feature FSL_FEATURE_PDM_HAS_DECIMATION_FILTER_BYPASS to config CTRL_2[DEC_BYPASS] field.
- Modify code to make the OSR value is not limited to 16.

[2.8.1]

- Improvements
- Added feature FSL_FEATURE_PDM_HAS_NO_DOZEN to handle nonexistent CTRL_1[DOZEN] field.

[2.8.0]

- Improvements
- Added feature FSL_FEATURE_PDM_HAS_NO_HWVAD to remove the support of hadware voice activity detector.
- Added feature FSL_FEATURE_PDM_HAS_NO_FILTER_BUFFER to remove the support of FIR_RDY bitfield in STAT register.

[2.7.4]

- Bug Fixes
  - Fixed driver can not determine the specific float number of clock divider.
  - Fixed PDM_ValidateSrcClockRate calculates PDM channel in wrong method issue.

[2.7.3]

- Improvements
- Added feature FSL_FEATURE_PDM_HAS_NO_VADEF to remove the support of VADEF bit-field in VAD0_STAT register.

[2.7.2]

- Improvements
- Added feature FSL_FEATURE_PDM_HAS_NO_MINIMUM_CLKDIV to decide whether the minimum clock frequency division is required.

[2.7.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 10.3, 10.1, 10.4, 14.4

[2.7.0]

- Improvements
  - Added api PDM_EnableHwvadInterruptCallback to support handle hwvad IRQ in PDM driver.
  - Corrected the sample rate configuration for non high quality mode.
  - Added api PDM_SetChannelGain to support adjust the channel gain.

[2.6.0]

- Improvements
  - Added new features FSL_FEATURE_PDM_HAS_STATUS_LOW_FREQ/FSL_FEATURE_PDM_HAS_DC_OUT

[2.5.0]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 16.5, 10.4, 10.3, 10.1, 11.9, 17.7, 10.6, 14.4, 11.8, 11.6.

[2.4.1]

- Bug Fixes
  - Fixed MDK 66-D warning in pdm driver.

[2.4.0]

- Improvements
  - Added api PDM_TransferSetChannelConfig/PDM_ReadFifo to support read different width data.
  - Added feature FSL_FEATURE_PDM_HAS_RANGE_CTRL and api PDM_ClearRangeStatus/PDM_GetRangeStatus for range register.
- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 14.4, 10.3, 10.4.

[2.3.0]

- Improvements
  - Enabled envelope/energy voice detect mode by adding apis PDM_SetHwvadInEnvelopeBasedMode/PDM_SetHwvadInEnergyBasedMode.
  - Added feature FSL_FEATURE_PDM_CHANNEL_NUM for different SOC.

[2.2.1]

- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 10.1, 10.3, 10.4, 10.6, 10.7, 11.3, 11.8, 14.4, 17.7, 18.4.
  - Added medium quality mode support in function PDM_SetSampleRateConfig.

[2.2.0]

- Improvements
  - Added api PDM_SetSampleRateConfig to improve user experience and marked api PDM_SetSampleRate as deprecated.

[2.1.1]

- Improvements
- Used new SDMA API SDMA_SetDoneConfig instead of SDMA_EnableSwDone for PDM SDMA driver.

[2.1.0]

- Improvements
  - Added software buffer queue for transactional API.

[2.0.1]

- Improvements
  - Improved HWVAD feature.

[2.0.0]

- Initial version.

PDM_EDMA

[2.6.5]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8.

[2.6.4]

- Improvements
  - Add handling for runtime change of number of linked transfers

[2.6.3]

- Improvements
  - Add EDMA ext API to accommodate more types of EDMA.

[2.6.2]

- Improvements
  - Add macro MCUX_SDK_PDM_EDMA_PDM_ENABLE_INTERNAL to let the user decide whether to enable it when calling PDM_TransferReceiveEDMA.

[2.6.1]

- Bug Fixes
  - Fixed violation of MISRA C-2012 Rule 10.3, 10.4.

[2.6.0]

- Improvements
  - Updated api PDM_TransferReceiveEDMA to support channel block interleave transfer.
  - Added new api PDM_TransferSetMultiChannelInterleaveType to support channel interleave type configurations.

[2.5.0]

- Refer PDM driver change log 2.1.0 to 2.5.0

PINT

[2.3.1]

- New Features
  - Added common IRQ handler entry PINT_DriverIRQHandler.

[2.3.0]

- Improvements
  - Add API PINT_EnableInterruptByIndex and PINT_DisableInterruptByIndex to provide more granular interrupt control.

[2.2.0]

- Fixed
  - Fixed the issue that clear interrupt flag when it's not handled. This causes events to be lost.
- Changed
  - Used one callback for one PINT instance. It's unnecessary to provide different callbacks for all PINT events.

[2.1.13]

- Improvements
  - Added instance array for PINT to adapt more devices.
  - Used release reset instead of reset PINT which may clear other related registers out of PINT.

[2.1.12]

- Bug Fixes
  - Fixed coverity issue.

[2.1.11]

- Bug Fixes
  - Fixed MISRA C-2012 rule 10.7 violation.

[2.1.10]

- New Features
  - Added the driver support for MCXN10 platform with combined interrupt handler.

[2.1.9]

- Bug Fixes
  - Fixed MISRA-2012 rule 8.4.

[2.1.8]

- Bug Fixes
  - Fixed MISRA-2012 rule 10.1 rule 10.4 rule 10.8 rule 18.1 rule 20.9.

[2.1.7]

- Improvements
  - Added fully support for the SECPINT, making it can be used just like PINT.

[2.1.6]

- Bug Fixes
  - Fixed the bug of not enabling common pint clock when enabling security pint clock.

[2.1.5]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - Fixed rule 10.1 rule 10.3 rule 10.4 rule 10.8 rule 14.4.
  - Changed interrupt init order to make pin interrupt configuration more reasonable.

[2.1.4]

- Improvements
  - Added feature to control distinguish PINT/SECPINT relevant interrupt/clock configurations for PINT_Init and PINT_Deinit API.
  - Swapped the order of clearing PIN interrupt status flag and clearing pending NVIC interrupt in PINT_EnableCallback and PINT_EnableCallbackByIndex function.
  - Bug Fixes
    - Fixed build issue caused by incorrect macro definitions.

[2.1.3]

- Bug fix:
  - Updated PINT_PinInterruptClrStatus to clear PINT interrupt status when the bit is asserted and check whether was triggered by edge-sensitive mode.
  - Write 1 to IST corresponding bit will clear interrupt status only in edge-sensitive mode and will switch the active level for this pin in level-sensitive mode.
  - Fixed MISRA c-2012 rule 10.1, rule 10.6, rule 10.7.
  - Added FSL_FEATURE_SECPINT_NUMBER_OF_CONNECTED_OUTPUTS to distinguish IRQ relevant array definitions for SECPINT/PINT on lpc55s69 board.
  - Fixed PINT driver c++ build error and remove index offset operation.

[2.1.2]

- Improvement:
  - Improved way of initialization for SECPINT/PINT in PINT_Init API.

[2.1.1]

- Improvement:
  - Enabled secure pint interrupt and add secure interrupt handle.

[2.1.0]

- Added PINT_EnableCallbackByIndex/PINT_DisableCallbackByIndex APIs to enable/disable callback by index.

[2.0.2]

- Added control macro to enable/disable the RESET and CLOCK code in current driver.

[2.0.1]

- Bug fix:
  - Updated PINT driver to clear interrupt only in Edge sensitive.

[2.0.0]

- Initial version.

PNGDEC

[2.0.1]

- Bug Fixes
  - Fixed CERT-C violations.

[2.0.0]

- Initial version.

POWER

[2.5.1]

- Improvement
  - Added configurable timeout support for endless while loop.

[2.5.0]

- New features
  - Added API POWER_EnableSleepAFBB to configure body bias to AFBB in deep sleep mode.
  - Added user configurable macro POWER_LDO_SAFE_MARGIN(x) with default value “0” to allow adding LDO output safe margin when using API POWER_ConfigRegulatorSetpointsForFreq.

[2.4.2]

- Bug fixes
  - Clear PDRUNCFG0 bit6 and bit8 before entering Deep Power Down(DPD) mode to make sure the domain is reset correctly after wake up from DPD.

[2.4.1]

- Bug fixes
  - Fixed bugs in POWER_DisableSleepRegulators POWER_EnableSleepRegulators.

[2.4.0]

- New features
  - Added APIs POWER_CalcVoltLevel, POWER_ConfigRegulatorSetpointsForFreq.

[2.3.0]

- Bug fixes
  - Fixed bugs in LDO Setpoint setting APIs.
  - Fixed bugs in XSPI recovery when XIP.
- New features
  - Added API POWER_EnterPowerMode.

[2.2.1]

- Bug fixes
  - Fixed bugs in enum _power_hwwake_src.

[2.2.0]

- New features
  - Added parameter checks in setpoint configuration APIs.
- Bug fixes
  - Fixed bugs in power mode entry API.

[2.1.0]

- New features
  - Added POWER_ConfigLvdSetpoints API.

[2.0.0]

- Initial version.

PUFv3

[2.0.4]

- Fix warning.

[2.0.3]

- Update for various PUF CTRL wrapper.

[2.0.2]

- Fix MISRA issue in driver.

[2.0.1]

- Fix PUF initialization issue and update driver to reflect SoC header changes.

[2.0.0]

- Initial version.

PVTS

[2.1.0]

- New features
- Added API PVTS_ReadParametersFromOTP to read A/B paramters of PVTS.

[2.0.2]

- New features
- Updated driver per ROMAPI changes.

[2.0.1]

- Improvement
  - Fixed MSG violations. Fixed MISRA C-2012 violations.

[2.0.0]

- Initial version.

RESET

[2.1.0]

- New features
  - Added RESET_SetFlashStateContext API.

[2.0.2]

- Bug fixes
  - Added FREQME_RSTS_N.

[2.0.1]

- Refined the GPIO_RSTS array so that the element number matches the instance index
- Added SDADC_RSTS.

[2.0.0]

- initial version.

ROMAPI

[2.1.0]

- New features
  - Updated OTP API to support Silicon B0.

[2.0.0]

- initial version.

SAI

[2.4.11]

- Bug Fixes
  - Fixed MSG findings for platforms with multiple FIFO size support.

[2.4.10]

- Improvements
  - Allow enabling/disabling implicit channel configuration.
  - Allow NULL FIFO watermark.
- Bug Fixes
  - Fix compilation warnings when asserts are disabled

[2.4.9]

- Added Errata ERR051421 workaround.

[2.4.8]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

[2.4.7]

- Added conditional support for bit clock swap feature
- Added common IRQ handler entry SAI_DriverIRQHandler.

[2.4.6]

- Bug Fixes
  - Fixed the IAR build warning.

[2.4.5]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

[2.4.4]

- Bug Fixes
  - Fixed enumeration sai_fifo_combine_t - add RX configuration.

[2.4.3]

- Bug Fixes
  - Fixed enumeration sai_fifo_combine_t value configuration issue.

[2.4.2]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.4.1]

- Bug Fixes
  - Fixed bitWidth incorrectly assigned issue.

[2.4.0]

- Improvements
  - Removed deprecated APIs.

[2.3.8]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4.

[2.3.7]

- Improvements
  - Change feature “FSL_FEATURE_SAI_FIFO_COUNT” to “FSL_FEATURE_SAI_HAS_FIFO”.
  - Added feature “FSL_FEATURE_SAI_FIFO_COUNTn(x)” to align SAI fifo count function with IP in function

[2.3.6]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 5.6.

[2.3.5]

- Improvements
  - Make driver to be aarch64 compatible.

[2.3.4]

- Bug Fixes
  - Corrected the fifo combine feature macro used in driver.

[2.3.3]

- Bug Fixes
  - Added bit clock polarity configuration when sai act as slave.
  - Fixed out of bound access coverity issue.
  - Fixed violations of MISRA C-2012 rule 10.3, 10.4.

[2.3.2]

- Bug Fixes
  - Corrected the frame sync configuration when sai act as slave.

[2.3.1]

- Bug Fixes
  - Corrected the peripheral name in function SAI0_DriverIRQHandler.
  - Fixed violations of MISRA C-2012 rule 17.7.

[2.3.0]

- Bug Fixes
  - Fixed the build error caused by the SOC has no fifo feature.

[2.2.3]

- Bug Fixes
  - Corrected the peripheral name in function SAI0_DriverIRQHandler.

[2.2.2]

- Bug Fixes
  - Fixed the issue of MISRA 2004 rule 9.3.
  - Fixed sign-compare warning.
  - Fixed the PA082 build warning.
  - Fixed sign-compare warning.
  - Fixed violations of MISRA C-2012 rule 10.3,17.7,10.4,8.4,10.7,10.8,14.4,17.7,11.6,10.1,10.6,8.4,14.3,16.4,18
  - Allow to reset Rx or Tx FIFO pointers only when Rx or Tx is disabled.
- Improvements
  - Added 24bit raw audio data width support in sai sdma driver.
  - Disabled the interrupt/DMA request in the SAI_Init to avoid generates unexpected sai FIFO requests.

[2.2.1]

- Improvements
  - Added mclk post divider support in function SAI_SetMasterClockDivider.
  - Removed useless configuration code in SAI_RxSetSerialDataConfig.
- Bug Fixes
  - Fixed the SAI SDMA driver build issue caused by the wrong structure member name used in the function SAI_TransferRxSetConfigSDMA/SAI_TransferTxSetConfigSDMA.
  - Fixed BAD BIT SHIFT OPERATION issue caused by the FSL_FEATURE_SAI_CHANNEL_COUNTn.
  - Applied ERR05144: not set FCONT = 1 when TMR > 0, otherwise the TX may not work.

[2.2.0]

- Improvements
  - Added new APIs for parameters collection and simplified user interfaces:
    - SAI_Init
    - SAI_SetMasterClockConfig
    - SAI_TxSetBitClockRate
    - SAI_TxSetSerialDataConfig
    - SAI_TxSetFrameSyncConfig
    - SAI_TxSetFifoConfig
    - SAI_TxSetBitclockConfig
    - SAI_TxSetConfig
    - SAI_TxSetTransferConfig
    - SAI_RxSetBitClockRate
    - SAI_RxSetSerialDataConfig
    - SAI_RxSetFrameSyncConfig
    - SAI_RxSetFifoConfig
    - SAI_RxSetBitclockConfig
    - SAI_RXSetConfig
    - SAI_RxSetTransferConfig
    - SAI_GetClassicI2SConfig
    - SAI_GetLeftJustifiedConfig
    - SAI_GetRightJustifiedConfig
    - SAI_GetTDMConfig

[2.1.9]

- Improvements
  - Improved SAI driver comment for clock polarity.
  - Added enumeration for SAI for sample inputs on different edges.

  - Changed FSL_FEATURE_SAI_CHANNEL_COUNT to FSL_FEATURE_SAI_CHANNEL_COUNTn(base) for the difference between the different SAI instances.
- Added new APIs:
  - SAI_TxSetBitClockDirection
  - SAI_RxSetBitClockDirection
  - SAI_RxSetFrameSyncDirection
  - SAI_TxSetFrameSyncDirection

[2.1.8]

- Improvements
  - Added feature macro test for the sync mode2 and mode 3.
  - Added feature macro test for masterClockHz in sai_transfer_format_t.

[2.1.7]

- Improvements
  - Added feature macro test for the mclkSource member in sai_config_t.
  - Changed "FSL_FEATURE_SAI5_SAI6_SHARE_IRQ" to "FSL_FEATURE_SAI_SAI5_SAI6_SHARE_IRQ".
  - Added #ifndef #endif check for SAI_XFER_QUEUE_SIZE to allow redefinition.
- Bug Fixes
  - Fixed build error caused by feature macro test for mclkSource.

[2.1.6]

- Improvements
  - Added feature macro test for mclkSourceClockHz check.
  - Added bit clock source name for general devices.
- Bug Fixes
  - Fixed incorrect channel numbers setting while calling RX/TX set format together.

[2.1.5]

- Bug Fixes
  - Corrected SAI3 driver IRQ handler name.
  - Added I2S4/5/6 IRQ handler.
  - Added base in handler structure to support different instances sharing one IRQ number.
- New Features
  - Updated SAI driver for MCR bit MICS.
  - Added 192 KHZ/384 KHZ in the sample rate enumeration.
  - Added multi FIFO interrupt/SDMA transfer support for TX/RX.
  - Added an API to read/write multi FIFO data in a blocking method.
  - Added bclk bypass support when bclk is same with mclk.

[2.1.4]

- New Features
  - Added an API to enable/disable auto FIFO error recovery in platforms that support this feature.
  - Added an API to set data packing feature in platforms which support this feature.

[2.1.3]

- New Features
  - Added feature to make I2S frame sync length configurable according to bitWidth.

[2.1.2]

- Bug Fixes
  - Added 24-bit support for SAI eDMA transfer. All data shall be 32 bits for send/receive, as eDMA cannot directly handle 3-Byte transfer.

[2.1.1]

- Improvements
  - Reduced code size while not using transactional API.

[2.1.0]

- Improvements
  - API name changes:
    * SAI_GetSendRemainingBytes -> SAI_GetSentCount.
    * SAI_GetReceiveRemainingBytes -> SAI_GetReceivedCount.
    * All names of transactional APIs were added with “Transfer” prefix.
    * All transactional APIs use base and handle as input parameter.
    * Unified the parameter names.
- Bug Fixes
  - Fixed WLC bug while reading TCSR/RCSR registers.
  - Fixed MOE enable flow issue. Moved MOE enable after MICS settings in SAI_TxInit/SAI_RxInit.

[2.0.0]

- Initial version.

SAI_EDMA

[2.7.4]

- Bug Fixes
  - Fixed MSG finding for platforms with multiple FIFO size support.

[2.7.3]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

[2.7.2]

- Improvements
  - Add macros MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL and MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL to let the user decide whether to enable SAI when calling SAI_TransferSendEDMA/SAI_TransferReceiveEDMA.

[2.7.1]

- Improvements
  - Add EDMA ext API to accommodate more types of EDMA.

[2.7.0]

- Improvements
  - Updated api SAI_TransferReceiveEDMA to support voice channel block interleave transfer.
  - Updated api SAI_TransferSendEDMA to support voice channel block interleave transfer.
  - Added new api SAI_TransferSetInterleaveType to support channel interleave type configurations.

[2.6.0]

- Improvements
  - Removed deprecated APIs.

[2.5.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 20.7.

[2.5.0]

- Improvements
  - Added new api SAI_TransferSendLoopEDMA/SAI_TransferReceiveLoopEDMA to support loop transfer.
  - Added multi sai channel transfer support.

[2.4.0]

- Improvements
  - Added new api SAI_TransferGetValidTransferSlotsEDMA which can be used to get valid transfer slot count in the sai edma transfer queue.
  - Deprecated the api SAI_TransferRxSetFormatEDMA and SAI_TransferTxSetFormatEDMA.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3,10.4.

[2.3.2]

- Refer SAI driver change log 2.1.0 to 2.3.2

SCTIMER

[2.6.1]

- Bug Fixes
  - Fixed CERT-C Array (CERT ARR30-C) violation issues in SCTIMER_FindSharedPeriodEvent and SCTIMER_UpdateComplementaryPwmDutycycle.
  - Fixed UNUSED_VALUE issue in SCTIMER_SetupPwm.

[2.6.0]

- New Features
  - Added SCTIMER_SetupSharedPeriodPwm() and SCTIMER_UpdatePwmDutycycleByEvent() for shared-period multi-channel PWM: same-frequency channels reuse one period/limit event (K channels cost K + 1 events instead of 2 * K), and duty updates take explicit period/pulse events with immediate or glitchless next-period modes.
  - Added SCTIMER_SetupComplementaryPwm() and SCTIMER_UpdateComplementaryPwmDutycycle() for complementary PWM output pairs with programmable dead time (edge- and center-aligned), including a shoot-through guard and shared-period reuse across pairs.
- Other Changes
  - Deprecated SCTIMER_SetupPwm() and SCTIMER_UpdatePwmDutycycle() in favor of SCTIMER_SetupSharedPeriodPwm() and SCTIMER_UpdatePwmDutycycleByEvent(); both remain functional with unchanged behavior.

[2.5.6]

- Bug Fixes
  - Fixed SCTIMER_CreateAndScheduleEvent setting EV[n].CTRL[HEVENT] only for match-based events. IO-only events (COMBMODE = 2) bound to Counter_H in 16-bit mode (UNIFY = 0) now correctly have HEVENT = 1, so their state mask is compared against STATE_H instead of STATE_L.

[2.5.5]

- Bug Fixes
  - Fixed CERT-C Array (CERT ARR30-C) violation issues.

[2.5.4]

- New Features
  - Added common IRQ handler entry SCTIMER_DriverIRQHandler.

[2.5.3]

- Bug Fixes
  - Fixed overflow error in SCT_EV_STATE_STATEMSKn macro definition.

[2.5.2]

- Bug Fixes
  - Fixed CERT INT31-C violation issues.

[2.5.1]

- Bug Fixes
  - Fixed bug in SCTIMER_SetupCaptureAction: When kSCTIMER_Counter_H is selected, events 12-15 and capture registers 12-15 CAPn_H field can't be used.

[2.5.0]

- Improvements
  - Add SCTIMER_GetCaptureValue API to get capture value in capture registers.

[2.4.9]

- Improvements
  - Supported platforms which don't have system level SCTIMER reset.

[2.4.8]

- Bug Fixes
  - Fixed the issue that the SCTIMER_UpdatePwmDutycycle() can't writes MATCH_H bit and RELOADn_H.

[2.4.7]

- Bug Fixes
  - Fixed the issue that the SCTIMER_UpdatePwmDutycycle() can't configure 100% duty cycle PWM.

[2.4.6]

- Bug Fixes
  - Fixed the issue where the H register was not written as a word along with the L register.
  - Fixed the issue that the SCTIMER_SetCOUNTValue() is not configured with high 16 bits in unify mode.

[2.4.5]

- Bug Fixes
  - Fix SCT_EV_STATE_STATEMSKn macro build error.

[2.4.4]

- Bug Fixes
  - Fix MISRA C-2012 issue 10.8.

[2.4.3]

- Bug Fixes
  - Fixed the wrong way of writing CAPCTRL and REGMODE registers in SCTIMER_SetupCaptureAction.

[2.4.2]

- Bug Fixes
  - Fixed SCTIMER_SetupPwm 100% duty cycle issue.

[2.4.1]

- Bug Fixes
  - Fixed the issue that MATCHn_H bit and RELOADn_H bit could not be written.

[2.4.0]

[2.3.0]

- Bug Fixes
  - Fixed the potential overflow issue of pulseperiod variable in SCTIMER_SetupPwm/SCTIMER_UpdatePwmDutycycle API.
  - Fixed the issue of SCTIMER_CreateAndScheduleEvent API does not correctly work with 32 bit unified counter.
  - Fixed the issue of position of clear counter operation in SCTIMER_Init API.
- Improvements
  - Update SCTIMER_SetupPwm/SCTIMER_UpdatePwmDutycycle to support generate 0% and 100% PWM signal.
  - Add SCTIMER_SetupEventActiveDirection API to configure event activity direction.

  - Update SCTIMER_StartTimer/SCTIMER_StopTimer API to support start/stop low counter and high counter at the same time.
  - Add SCTIMER_SetCounterState/SCTIMER_GetCounterState API to write/read counter current state value.
  - Update APIs to make it meaningful.
    * SCTIMER_SetEventInState
    * SCTIMER_ClearEventInState
    * SCTIMER_GetEventInState

[2.2.0]

- Improvements
  - Updated for 16-bit register access.

[2.1.3]

- Bug Fixes
  - Fixed the issue of uninitialized variables in SCTIMER_SetupPwm.
  - Fixed the issue that the Low 16-bit and high 16-bit work independently in SCTIMER driver.
- Improvements
  - Added an enumerable macro of unify counter for user.
    * kSCTIMER_Counter_U
  - Created new APIs for the RTC driver.
    * SCTIMER_SetupStateLdMethodAction
    * SCTIMER_SetupNextStateActionwithLdMethod
    * SCTIMER_SetCOUNTValue
    * SCTIMER_GetCOUNTValue
    * SCTIMER_SetEventInState
    * SCTIMER_ClearEventInState
    * SCTIMER_GetEventInState
  - Deprecated legacy APIs for the RTC driver.
    * SCTIMER_SetupNextStateAction

[2.1.2]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.3, 10.4, 10.6, 10.7, 11.9, 14.2 and 15.5.

[2.1.1]

- Improvements
  - Updated the register and macro names to align with the header of devices.

[2.1.0]

- Bug Fixes
  - Fixed issue where SCT application level Interrupt handler function is occupied by SCT driver.
  - Fixed issue where wrong value for INSYNC field inside SCTIMER_Init function.
  - Fixed issue to change Default value for INSYNC field inside SCTIMER_GetDefaultConfig.

[2.0.1]

- New Features
  - Added control macro to enable/disable the RESET and CLOCK code in current driver.

[2.0.0]

- Initial version.

SDADC

[2.0.1]

- Improvements
  - Fixed MISRA C-2012 issues.

[2.0.0]

- Initial version.

SEMA42

[2.1.2]

- Improvements
  - Fix MISRA C-2012 Rule 20.8 violations.

[2.1.1]

- Improvements
  - Updated SEMA42_TryLock function to avoid unsigned integer operations wrap issue.

[2.1.0]

- New Features
  - Added SEMA42_BUSY_POLL_COUNT parameter to prevent infinite polling loops in SEMA42 operations.
  - Added timeout mechanism to all polling loops in SEMA42 driver code.
- Improvements
  - Updated SEMA42_Lock function to return status_t instead of void for better error handling.

  - Enhanced documentation to clarify timeout behavior and return values.

[2.0.4]

- Improvements
  - Release peripheral from reset if necessary in init function.

[2.0.3]

- Improvements
  - Changed to implement SEMA42_Lock base on SEMA42_TryLock.

[2.0.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 17.7.

[2.0.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 14.4, 18.1.

[2.0.0]

- Initial version.

SYSPM

[2.3.1]

- Improvements
  - Add timeout for while loop.

[2.3.0]

- New Features
  - Supported instruction counter.

[2.2.0]

- New Features
  - Supported platforms which only have one monitor.
  - Supported platforms whose instrction counter can't be cleared.
  - Supported platforms whose counter can't be disabled.

[2.1.0]

- New Features
  - Added new APIs SYSPM_Init and SYSPM_Deinit.

[2.0.0]

- Initial version.

TRNG

[2.0.23]

- New features:
  - Updated TRNG default values for MCXA577 devices based on silicon characterization.

[2.0.22]

- New features:
  - Added support for KW43 and MCXW70 devices.

[2.0.21]

- New features:
  - Added support for MCXC devices.

[2.0.20]

- New features:
  - Added support for MCXA devices.

[2.0.19]

- New features:
  - Added support for MCXA and MCXL.

[2.0.18]

- Bug fix:
  - TRNG health checks now done in software on RT5xx and RT6xx.

[2.0.17]

- New features:
  - Add support for RT700.

[2.0.16]

- Improvements:
  - Added support for Dual oscillator mode.

[2.0.15]

- Other changes:
  - Changed TRNG_USER_CONFIG_DEFAULT_XXX values according to latest reccomended by design team.

[2.0.14]

- New features:
  - Add support for RW610 and RW612.

[2.0.13]

- Bug fix:
  - After deepsleep it might return error, added clearing bits in TRNG_GetRandomData() and generating new entropy.
  - Modified reloading entropy in TRNG_GetRandomData(), for some data length it doesn't reloading entropy correctly.

[2.0.12]

- Bug fix:
  - For KW34A4_SERIES, KW35A4_SERIES, KW36A4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.

[2.0.11]

- Bug fix:
  - Add clearing pending errors in TRNG_Init().

[2.0.10]

- Bug Fix:
  - Fixed doxygen issues.

[2.0.9]

- Bug Fix:
  - Fix HIS_CCM metrics issues.

[2.0.8]

- Bug fix:
  - For K32L2A41A_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv4.

[2.0.7]

- Bug fix:
  - Fix MISRA 2004 issue rule 12.5.

[2.0.6]

- Bug fix:
  - For KW35Z4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.

[2.0.5]

- Improvements:
  - For FRQMIN, FRQMAX and OSCDIV, add possibility to use device specific preprocessor macro to define default value in TRNG user configuration structure.

[2.0.4]

- Bug Fix:
  - Fix MISRA-2012 issues.
    - Rule 10.1, rule 10.3, rule 13.5, rule 16.1.

[2.0.3]

- Improvements:
  - update TRNG_Init to restart new entropy generation.

[2.0.2]

- Improvements:
  - fix MISRA issues
    - Rule 14.4.

[2.0.1]

- New features:
  - Set default OSCDIV for Kinetis devices KL8x and KL28Z.
- Other changes:
  - Changed default OSCDIV for K81 to divide by 2.

[2.0.0]

- Initial version.

USDHC

[2.8.10]

- Bug Fixes
  - Fixed some quality issues.

[2.8.9]

- Bug Fixes
  - Fixed some quality issues.
  - Do not invalidate cache on non-blocking non-DMA transfer.
- New Features
  - Added common IRQ handler entry USDHC_DriverIRQHandler.

[2.8.8]

- Bug Fixes
  - Fixed build issue with armgcc O3.

[2.8.7]

- Bug Fixes
  - Disabled CMD error check for standard tuning per RM.

[2.8.6]

- Bug Fixes
  - Invalidate cache after blocking read.

[2.8.5]

- Improvements
  - Enable the driver to be AARCH64 compatible.

[2.8.4]

- Improvements
  - Add feature macro FSL_FEATURE_USDHC_HAS_NO_VS18.

[2.8.3]

- Improvements
  - Improved api USDHC_EnableAutoTuningForCmdAndData to adapt to new bit field name for USDHC_VEND_SPEC2 register.

[2.8.2]

- Improvements
  - Added feature macro FSL_FEATURE_USDHC_HAS_NO_VOLTAGE_SELECT.

[2.8.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9.

[2.8.0]

- Improvements
  - Fixed the mmc boot transfer failed issue which is caused by the Dma complete interrupt not enabled.
  - Marked api USDHC_AdjustDelayForManualTuning as deprecated and added new api USDHC_SetTuingDelay/USDHC_GetTuningDelayStatus.
  - Improved the manual tuning flow accroding to specification.
  - Added memory address conversion to support buffers which could only be accessed using alias address by non-core masters.
  - Fixed violations of MISRA C-2012 rule 10.4.

[2.7.0]

- Improvements
  - Added api USDHC_TransferScatterGatherADMANonBlocking to support scatter gather transfer.
  - Added feature FSL_FEATURE_USDHC_REGISTER_HOST_CTRL_CAP_HAS_NO_RETUNING_TIME_COUNT for re-tuning time counter field in HOST_CTRL_CAP register.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9, 10.1, 10.3, 10.4, 8.4.

[2.6.0]

- Improvements
  - Added api USDHC_SetStandardTuningCounter to support adjust tuning counter of Standard tuning.

[2.5.1]

- Improvements
  - Used different status code for command and data interrupt callback.
  - Added cache line invalidate for receive buffer in driver IRQ handler to fix CM7 speculative access issue.

[2.5.0]

- Improvements
  - Added new api USDHC_SetStrobeDllOverride for HS400 strobe dll override mode delay taps configurations.
  - Corrected the STROBE DLL configurations sequence.

[2.4.0]

- Improvements
  - Added feature macro for read/write burst length.
    - Disabled redundant interrupt per different transfer request.

        * Disabled interrupt and reset command/data pointer in handle when transfer completes.

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 11.9, 15.7, 4.7, 16.4, 10.1, 10.3, 10.4, 11.3, 14.4, 10.6, 17.7, 16.1, 16.3.
  - Fixed PA082 build warning.
  - Fixed logically dead code Coverity issue.

[2.3.0]

- Improvements
  - Added USDHC_SetDataConfig API to support manual tuning.
  - Removed the limitaion that source clock must be bigger than the target in function USDHC_SetSdClock by using source clock frequency as target directly.
  - Added peripheral reset in USDHC_Init function.
  - Added tuning reset support in function USDHC_Reset function.

[2.2.8]

- Bug Fixes
  - Fixed out-of bounds write in function USDHC_ReceiveCommandResponse.

[2.2.7]

- Improvements
  - Added API USDHC_GetEnabledInterruptStatusFlags and used in USDHC_TransferHandleIRQ.
  - Removed useless member interruptFlags in usdhc_handle_t.

[2.2.6]

- Improvements
  - Added address align check for ADMA descriptor table address.
  - Changed USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY to (65536-4096) to make sure the data address is 4KB align for a transfer which need more than one ADMA1 descriptor.

[2.2.5]

- Bug Fixes
  - Fixed MDK 66-D warning.

[2.2.4]

- Bug Fixes
  - Fixed issue that real clock frequency wss mismatched with target clock frequency, which was caused by an incorrect prescaler calculation.
- New Features
  - Added control macro to enable/disable the CLOCK code in current driver.

[2.2.3]

- Bug Fixes
  - Fixed issue where AMDA did not disable with DMAEN clear.
- Improvements
  - Improved set clock function to check the output frequency range.
  - Dynamic set SDCLKFS during DDR enable or disable.

[2.2.2]

- Improvements
  - Improved read transfer cache maintain operation, combined clean, and invalidated them into one function.

[2.2.1]

- Bug Fixes
  - Disabled the invalidate cache operation for tuning.

[2.2.0]

- Improvements
  - Improved USDHC to support MMC boot feature.

[2.1.3]

- Bug Fixes
  - Fixed MISRA issue.

[2.1.2]

- Bug Fixes
  - Fixed Coverity issue.
  - Added base address and userData parameter for all callback functions.

[2.1.1]

- Improvements
  - Added cache maintain operation.
  - Added timeout status check for the DATA transfer which ignore error.
  - Added feature macro for SDR50/SDR104 mode.
  - Removed useless IRQ handler from different platforms.

[2.1.0]

- Improvements
  - Integrated tuning into transfer function.
  - Added strobe DLL feature.
  - Added enableAutoCommand23 in data structure.
  - Removed enable card clock function because the controller would handle the clock on/off.

[2.0.0]

- Initial version.

UTICK

[2.0.6]

- New Features
  - Added common IRQ handler entry UTICK_CommonDriverIRQHandler.

[2.0.5]

- Improvements
  - Improved for SOC RW610.

[2.0.4]

- Bug Fixes
  - Fixed compile fail issue of no-supporting PD configuration in utick driver.

[2.0.3]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules: 8.4, 14.4, 17.7

[2.0.2]

- Added new feature definition macro to enable/disable power control in drivers for some devices have no power control function.

[2.0.1]

- Added control macro to enable/disable the CLOCK code in current driver.

[2.0.0]

- Initial version.

WWDT

[2.1.11]

- Bug Fixes
  - Fixed CERT INT31-C static analysis finding by clamping boolean configuration values (WDEN/WDRESET/LOCK) to 0/1 before writing the MOD register in WWDT_Init.

[2.1.10]

- Bug Fixes
  - Chek WWDT_RSTS instead of FSL_FEATURE_WWDT_HAS_NO_RESET to determine whether the peripheral can be reset.

[2.1.9]

- Bug Fixes
  - Fixed violation of the MISRA C-2012 rule 10.4.

[2.1.8]

- Improvements
  - Updated the “WWDT_Init” API to add wait operation. Which can avoid the TV value read by CPU still be 0xFF (reset value) after WWDT_Init function returns.

[2.1.7]

- Bug Fixes
  - Fixed the issue that the watchdog reset event affected the system from PMC.
  - Fixed the issue of setting watchdog WDPROTECT field without considering the backwards compatibility.
  - Fixed the issue of clearing bit fields by mistake in the function of WWDT_ClearStatusFlags.

[2.1.5]

- Bug Fixes
  - deprecated a unusable API in WWWDT driver.
    * WWDT_Disable

[2.1.4]

- Bug Fixes
  - Fixed violation of the MISRA C-2012 rules Rule 10.1, 10.3, 10.4 and 11.9.
  - Fixed the issue of the inseparable process interrupted by other interrupt source.
    * WWDT_Init

[2.1.3]

- Bug Fixes
  - Fixed legacy issue when initializing the MOD register.

[2.1.2]

- Improvements
  - Updated the “WWDT_ClearStatusFlags” API and “WWDT_GetStatusFlags” API to match QN9090. WDTOF is not set in case of WD reset. Get info from PMC instead.

[2.1.1]

- New Features
  - Added new feature definition macro for devices which have no LCOK control bit in MOD register.
  - Implemented delay/retry in WWDT driver.

[2.1.0]

- Improvements
  - Added new parameter in configuration when initializing WWDT module. This parameter, which must be set, allows the user to deliver the WWDT clock frequency.

[2.0.0]

- Initial version.

XSPI

[2.7.7]

- Bug Fixes
  - Fixed CID 51973698 NO_EFFECT in XSPI_UpdateSFPConfig() where the non-EENV FRAD word2/word3 update was redundantly executed once per target group instead of once per FRAD region.

[2.7.6]

- Bug Fixes
  - Fixed CERT-C INT31-C violations in XSPI_TG_REG_ADDR_xxx macros

[2.7.5]

- Bug Fixes
  - Fixed CERT-C INT30-C violations in XSPI_WriteBlockingTG and XSPI_CommonIRQHandler.
  - Fixed XSPI1_0–XSPI1_4 IRQ handlers to route through XSPI_DriverIRQHandler with proper instance and envId bounds validation.

[2.7.4]

- New Features
  - Added common IRQ handler entry XSPI_DriverIRQHandler.

[2.7.3]

- Bug Fixes
  - Fixed CERT-C INT30-C and ARR30-C violations in XSPI_TransferBlocking, XSPI_WriteBlockingTG, XSPI_ReadBlockingTG, and XSPI_IRQHandler.

[2.7.2]

- Bug Fixes
  - Fixed typo where ptrSubBuffer2Config incorrectly used XSPI_BUFCR_SUBBUF1_DIV instead of XSPI_BUFCR_SUBBUF2_DIV.

[2.7.1]

- Bug Fixes
  - TGIPCRS[VLD] bit polling fix for cortex-m33 core1 in NETCMIX on i.MX943 Skip checking the TGIPCRS[VLD] bit to resolve an issue where the bit transition from 0 to 1 is extremely short, causing the master to read 0 and poll indefinitely. This issue occurs when the master is Cortex-M33 Core1 in NETCMIX on i.MX943. Note: The bit functions correctly when the master is Cortex-M7 Core0 in M7MIX0 or Cortex-M7 Core1 in M7MIX1.

[2.7.0]

- New Features
  - Added EENV support to setup XSPI based on diffferent target groups.
- Bug Fixes
  - Fixed issue of XSPI_UpdateSFPConfig() function. Disabled all global MDAD and FRAD setting before updating.

[2.6.1]

- Improvements
  - Support for macro to place functions in RAM. Functions of file needs to be place in RAM while using XSPI1 for RT700.
  - Support added to place variables in RAM. Its required when code from ram is using these variables.

- Bug Fixes
  - Add explicit boolean to integer conversions (? 1UL : 0UL) for enable flags in DLL configuration, data learning, device config, and AHB buffer settings.
  - Add assertions to prevent unsigned integer overflow in address increment operations.
  - Add assertions to prevent unsigned integer underflow in size subtraction operations.
  - Add safe type conversion with temporary variables for uint64_t to uint32_t operations in interrupt mask handling.
  - Add bounds checking for array access and enum value validation.
  - Add assertions for narrowing conversions in byte extraction operations.
  - Follow INT30-C and INT31-C CERT coding standards for safe integer operations.

[2.6.0]

- Improvements
  - Added feature macro support for devices with reserved register bit fields to improve compatibility across different device variants.
    - FSL_FEATURE_XSPI_HAS_DQS_LAT_EN
    - FSL_FEATURE_XSPI_HAS_END_CFG
    - FSL_FEATURE_XSPI_HAS_DOZE_MODE
  - Enhanced register access handling using conditional compilation to accommodate hardware differences.
  - Added supports of MAIN_XSPI0_DriverIRQHandler() and MAIN_XSPI1_DriverIRQHandler.
- New Features
  - Added AHB write terminate functionality support for devices with FSL_FEATURE_XSPI_HAS_WRTER_EN_BIT.
    - New API: XSPI_EnableAhbWriteTerminate() to control AHB write termination of read-prefetch operations.
    - New configuration field: enableWriteTerminate in xspi_ahb_access_config_t structure.

[2.5.2]

- Improvements
  - Fixed an issue with incorrect cache range address calculation. It caused issue when the cache operation size is equal to a multiple of the cache line size + 1.

[2.5.1]

- Improvements
  - Updated default value of sfpArbitrationLockTimeoutValue and ipAccessTimeoutValue to 0xFFFFFFFFUL.
  - Added #if defined(CACHE64_CTRL0_BASE) ... #endif section to support some devices that do not support CACHE64.

[2.5.0]

- Improvements
  - Updated XSPI_TransferBlocking() to support use case the transfersize bigger than page size.
  - Updated xspi_device_interface_type_t to support changes of page size for hyperram interface.
  - Updated XSPI_ReadBlocking() to fix an potential issue which cause TIMEOUT error.

[2.4.0]

- New Features
  - Added functions of control cache64.

[2.3.0]

- Improvements
  - Added new interface: XSPI_StartIpAccessNonBlocking();
  - Provied driver Irq handler for xspi driver;
  - Use ERRSTAT[ARB_WIN] to replace FSMSTAT[VLD].

[2.2.1]

- Improvements
  - Moved some frequently used variables(s_tgSfarsRegOffset, s_tgIpcrsRegOffset, s_sfpTgIpcrRegOffset, s_sfpTgIpSfarRegOffset, s_tgMdadRegOffset) to common code to offload of stack.
  - Added return status in case of timeout flag asserted during IP read access.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rules.

[2.2.0]

- Improvements
  - Improved xspi_device_config_t structure, removed some not-device related members.
  - Improved xspi_config_t structure, removed some device related settings.
  - Renamed XSPI_SetFlashConfig() to XSPI_SetDeviceConfig().
  - Decoupled settings of IP access, AHB access.
  - Added low-level interfaces to support IP access(including SFP), and AHB access(including performance monitor).

[2.1.0]

- Bug Fixes
  - Fixed issue of XSPI_Init() function, set MCR[MDIS] to disable clocks before settings of MCR register.
  - Unused deviceConfig in XSPI_GetDefaultConfig() function.

- Improvements
  - Updated structure for dllconfig, added method to allow user custom dll parameters.
  - Updated XSPI_UpdateDllValue() function to align with changes of dllconfig.
  - Move device specific value to device feature file.

[2.0.1]

- Bug Fixes
  - Fixed the XSPI DLL function.
  - Added ALIGN and X16Mode macro definition in driver.
  - updated XSPI PSRAM example.

[2.0.0]

- Initial version.

XSPI EDMA Driver

[2.0.4]

- Bug Fixes
  - Fixed violations in XSPI_TransferEDMACallback() function, avoid warning of unused parameters.

[2.0.3]

- Bug Fixes
  - Fixed violations of MSG.

[2.0.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rules.

[2.0.1]

- Improvements
  - Invoked new defined interface of xspi driver.
  - In callback function, disable RX/TX DMA.

[2.0.0]

- Initial version.

1.8 Driver API Reference Manual

This section provides a link to the Driver API RM, detailing available drivers and their usage to help you integrate hardware efficiently.

MIMXRT798S

1.9 Middleware Documentation

Find links to detailed middleware documentation for key components. While not all onboard middleware is covered, this serves as a useful reference for configuration and development.

1.9.1 VG-Lite GPU Library

VGLite Graphics Driver

1.9.2 Multicore

Multicore SDK

1.9.3 MCU Boot

mcuboot_opensource

1.9.4 eIQ

eiq

1.9.5 FreeMASTER

freemaster

1.9.6 NXP Wi-Fi

Wi-Fi, Bluetooth, 802.15.4

1.9.7 FreeRTOS

FreeRTOS

1.9.8 lwIP

lwIP

1.9.9 File systemFatfs

FatFs

1.9.10 DSP Audio Streamer

Xtensa Audio Framework (XAF)

1.9.11 Trusted-Frimware-M

Trusted-Firmware-M

Chapter 2

MIMXRT798S

2.1 ACMP: Analog Comparator Driver

void ACMP_Init(CMP_Type *base, const *acmp_config_t* *config)

Initializes the ACMP.

The default configuration can be got by calling *ACMP_GetDefaultConfig()*.

Parameters

- base – ACMP peripheral base address.
- config – Pointer to ACMP configuration structure.

void ACMP_Deinit(CMP_Type *base)

Deinitializes the ACMP.

Parameters

- base – ACMP peripheral base address.

void ACMP_GetDefaultConfig(*acmp_config_t* *config)

Gets the default configuration for ACMP.

This function initializes the user configuration structure to default value. The default value are:

Example:

```
config->enableHighSpeed = false;
config->enableInvertOutput = false;
config->useUnfilteredOutput = false;
config->enablePinOut = false;
config->enableHysteresisBothDirections = false;
config->hysteresisMode = kACMP_hysteresisMode0;
```

Parameters

- config – Pointer to ACMP configuration structure.

void ACMP_Enable(CMP_Type *base, bool enable)

Enables or disables the ACMP.

Parameters

- base – ACMP peripheral base address.
- enable – True to enable the ACMP.

void ACMP_EnableLinkToDAC(CMP_Type *base, bool enable)

Enables the link from CMP to DAC enable.

When this bit is set, the DAC enable/disable is controlled by the bit CMP_C0[EN] instead of CMP_C1[DACEN].

Parameters

- base – ACMP peripheral base address.
- enable – Enable the feature or not.

void ACMP_SetChannelConfig(CMP_Type *base, const *acmp_channel_config_t* *config)

Sets the channel configuration.

Note that the plus/minus mux's setting is only valid when the positive/negative port's input isn't from DAC but from channel mux.

Example:

```
acmp_channel_config_t configStruct = {0};
configStruct.positivePortInput = kACMP_PortInputFromDAC;
configStruct.negativePortInput = kACMP_PortInputFromMux;
configStruct.minusMuxInput = 1U;
ACMP_SetChannelConfig(CMP0, &configStruct);
```

Parameters

- base – ACMP peripheral base address.
- config – Pointer to channel configuration structure.

void ACMP_EnableDMA(CMP_Type *base, bool enable)

Enables or disables DMA.

Parameters

- base – ACMP peripheral base address.
- enable – True to enable DMA.

void ACMP_SetFilterConfig(CMP_Type *base, const *acmp_filter_config_t* *config)

Configures the filter.

The filter can be enabled when the filter count is bigger than 1, the filter period is greater than 0 and the sample clock is from divided bus clock or the filter is bigger than 1 and the sample clock is from external clock. Detailed usage can be got from the reference manual.

Example:

```
acmp_filter_config_t configStruct = {0};
configStruct.filterCount = 5U;
configStruct.filterPeriod = 200U;
configStruct.enableSample = false;
ACMP_SetFilterConfig(CMP0, &configStruct);
```

Parameters

- base – ACMP peripheral base address.
- config – Pointer to filter configuration structure.

void ACMP_SetDACConfig(CMP_Type *base, const *acmp_dac_config_t* *config)

Configures the internal DAC.

Example:

```
acmp_dac_config_t configStruct = {0};
configStruct.referenceVoltageSource = kACMP_VrefSourceVin1;
configStruct.DACValue = 20U;
configStruct.enableOutput = false;
configStruct.workMode = kACMP_DACWorkLowSpeedMode;
ACMP_SetDACConfig(CMP0, &configStruct);
```

Parameters

- base – ACMP peripheral base address.
- config – Pointer to DAC configuration structure. "NULL" is for disabling the feature.

void ACMP_EnableInterrupts(CMP_Type *base, uint32_t mask)

Enables interrupts.

Parameters

- base – ACMP peripheral base address.
- mask – Interrupts mask. See "_acmp_interrupt_enable".

void ACMP_DisableInterrupts(CMP_Type *base, uint32_t mask)

Disables interrupts.

Parameters

- base – ACMP peripheral base address.
- mask – Interrupts mask. See "_acmp_interrupt_enable".

uint32_t ACMP_GetStatusFlags(CMP_Type *base)

Gets status flags.

Parameters

- base – ACMP peripheral base address.

Returns

Status flags asserted mask. See "_acmp_status_flags".

void ACMP_ClearStatusFlags(CMP_Type *base, uint32_t mask)

Clears status flags.

Parameters

- base – ACMP peripheral base address.
- mask – Status flags mask. See "_acmp_status_flags".

void ACMP_SetDiscreteModeConfig(CMP_Type *base, const *acmp_discrete_mode_config_t* *config)

Configure the discrete mode.

Configure the discrete mode when supporting 3V domain with 1.8V core.

Parameters

- base – ACMP peripheral base address.
- config – Pointer to configuration structure. See "acmp_discrete_mode_config_t".

void ACMP_GetDefaultDiscreteModeConfig(*acmp_discrete_mode_config_t* *config)

Get the default configuration for discrete mode setting.

Parameters

- config – Pointer to configuration structure to be restored with the setting values.

FSL_ACMP_DRIVER_VERSION

ACMP driver version 2.5.0.

enum _acmp_interrupt_enable

Interrupt enable/disable mask.

Values:

enumerator kACMP_OutputRisingInterruptEnable

Enable the interrupt when comparator outputs rising.

enumerator kACMP_OutputFallingInterruptEnable

Enable the interrupt when comparator outputs falling.

enum _acmp_status_flags

Status flag mask.

Values:

enumerator kACMP_OutputRisingEventFlag

Rising-edge on compare output has occurred.

enumerator kACMP_OutputFallingEventFlag

Falling-edge on compare output has occurred.

enumerator kACMP_OutputAssertEventFlag

Return the current value of the analog comparator output.

enum _acmp_offset_mode

Comparator hard block offset control.

If OFFSET level is 1, then there is no hysteresis in the case of positive port input crossing negative port input in the positive direction (or negative port input crossing positive port input in the negative direction). Hysteresis still exists for positive port input crossing negative port input in the falling direction. If OFFSET level is 0, then the hysteresis selected by acmp_hysteresis_mode_t is valid for both directions.

Values:

enumerator kACMP_OffsetLevel0

The comparator hard block output has level 0 offset internally.

enumerator kACMP_OffsetLevel1

The comparator hard block output has level 1 offset internally.

enum _acmp_hysteresis_mode

Comparator hard block hysteresis control.

See chip data sheet to get the actual hysteresis value with each level.

Values:

enumerator kACMP_HysteresisLevel0
Offset is level 0 and Hysteresis is level 0.

enumerator kACMP_HysteresisLevel1
Offset is level 0 and Hysteresis is level 1.

enumerator kACMP_HysteresisLevel2
Offset is level 0 and Hysteresis is level 2.

enumerator kACMP_HysteresisLevel3
Offset is level 0 and Hysteresis is level 3.

enum _acmp_reference_voltage_source
CMP Voltage Reference source.

Values:

enumerator kACMP_VrefSourceVin1
Vin1 is selected as resistor ladder network supply reference Vin.

enumerator kACMP_VrefSourceVin2
Vin2 is selected as resistor ladder network supply reference Vin.

enum _acmp_port_input
Port input source.

Values:

enumerator kACMP_PortInputFromDAC
Port input from the 8-bit DAC output.

enumerator kACMP_PortInputFromMux
Port input from the analog 8-1 mux.

enum _acmp_dac_work_mode
Internal DAC's work mode.

Values:

enumerator kACMP_DACWorkLowSpeedMode
DAC is selected to work in low speed and low power mode.

enumerator kACMP_DACWorkHighSpeedMode
DAC is selected to work in high speed high power mode.

typedef enum *_acmp_offset_mode* acmp_offset_mode_t
Comparator hard block offset control.

If OFFSET level is 1, then there is no hysteresis in the case of positive port input crossing negative port input in the positive direction (or negative port input crossing positive port input in the negative direction). Hysteresis still exists for positive port input crossing negative port input in the falling direction. If OFFSET level is 0, then the hysteresis selected by acmp_hysteresis_mode_t is valid for both directions.

typedef enum *_acmp_hysteresis_mode* acmp_hysteresis_mode_t

Comparator hard block hysteresis control.

See chip data sheet to get the actual hysteresis value with each level.

typedef enum *_acmp_reference_voltage_source* acmp_reference_voltage_source_t

CMP Voltage Reference source.

typedef enum *_acmp_port_input* acmp_port_input_t

Port input source.

typedef enum *_acmp_dac_work_mode* acmp_dac_work_mode_t

Internal DAC's work mode.

typedef struct *_acmp_config* acmp_config_t

Configuration for ACMP.

typedef struct *_acmp_channel_config* acmp_channel_config_t

Configuration for channel.

The comparator's port can be input from channel mux or DAC. If port input is from channel mux, detailed channel number for the mux should be configured.

typedef struct *_acmp_filter_config* acmp_filter_config_t

Configuration for filter.

typedef struct *_acmp_dac_config* acmp_dac_config_t

Configuration for DAC.

typedef struct *_acmp_discrete_mode_config* acmp_discrete_mode_config_t

Configuration for discrete mode.

CMP_C0_CFx_MASK

The mask of status flags cleared by writing 1.

struct _acmp_config

#include <fsl_acmp.h> Configuration for ACMP.

Public Members

acmp_offset_mode_t offsetMode

Offset mode.

acmp_hysteresis_mode_t hysteresisMode

Hysteresis mode.

bool enableHighSpeed

Enable High Speed (HS) comparison mode.

bool enableInvertOutput

Enable inverted comparator output.

bool useUnfilteredOutput

Set compare output(COUT) to equal COUTA(true) or COUT(false).

bool enablePinOut

The comparator output is available on the associated pin.

struct _acmp_channel_config

#include <fsl_acmp.h> Configuration for channel.

The comparator's port can be input from channel mux or DAC. If port input is from channel mux, detailed channel number for the mux should be configured.

Public Members

acmp_port_input_t positivePortInput

Input source of the comparator's positive port.

uint32_t plusMuxInput

Plus mux input channel(0~7).

acmp_port_input_t negativePortInput

Input source of the comparator's negative port.

uint32_t minusMuxInput

Minus mux input channel(0~7).

struct _acmp_filter_config

#include <fsl_acmp.h> Configuration for filter.

Public Members

uint32_t filterCount

Filter Sample Count. Available range is 1-7, 0 would cause the filter disabled.

uint32_t filterPeriod

Filter Sample Period. The divider to bus clock. Available range is 0-255.

struct _acmp_dac_config

#include <fsl_acmp.h> Configuration for DAC.

Public Members

acmp_reference_voltage_source_t referenceVoltageSource
Supply voltage reference source.

uint32_t DACValue
Value for DAC Output Voltage. Available range is 0-255.

bool enableOutput
Enable the DAC output.

struct _acmp_discrete_mode_config
#include <fsl_acmp.h> Configuration for discrete mode.

Public Members

bool enablePositiveChannelDiscreteMode
Positive Channel Continuous Mode Enable. By default, the continuous mode is used.

bool enableNegativeChannelDiscreteMode
Negative Channel Continuous Mode Enable. By default, the continuous mode is used.

2.2 CACHE: CACHE Memory Controller

uint32_t CACHE64_GetInstance(CACHE64_POLSEL_Type *base)
Returns an instance number given peripheral base address.

Parameters

- base – The peripheral base address.

Returns
CACHE64_POLSEL instance number starting from 0.

uint32_t CACHE64_GetInstanceByAddr(uint32_t address)
brief Returns an instance number given physical memory address.

param address The physical memory address.

Returns
CACHE64_CTRL instance number starting from 0.

status_t CACHE64_Init(CACHE64_POLSEL_Type *base, const *cache64_config_t* *config)
Initializes an CACHE64 instance with the user configuration structure.

This function configures the CACHE64 module with user-defined settings. Call the *CACHE64_GetDefaultConfig()* function to configure the configuration structure and get the default configuration.

Parameters

- base – CACHE64_POLSEL peripheral base address.
- config – Pointer to a user-defined configuration structure.

Return values
kStatus_Success – CACHE64 initialize succeed

void CACHE64_GetDefaultConfig(*cache64_config_t* *config)

Gets the default configuration structure.

This function initializes the CACHE64 configuration structure to a default value. The default values are first region covers whole cacheable area, and policy set to write back.

Parameters

- config – Pointer to a configuration structure.

void CACHE64_EnableCache(CACHE64_CTRL_Type *base)

Enables the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_DisableCache(CACHE64_CTRL_Type *base)

Disables the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_InvalidateCache(CACHE64_CTRL_Type *base)

Invalidates the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_InvalidateCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates cache by range.

Note: Address and size should be aligned to "CACHE64_LINESIZE_BYTE". The startAddr here will be forced to align to CACHE64_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be invalidated, should be larger than 0.

void CACHE64_CleanCache(CACHE64_CTRL_Type *base)

Cleans the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_CleanCacheByRange(uint32_t address, uint32_t size_byte)

Cleans cache by range.

Note: Address and size should be aligned to "CACHE64_LINESIZE_BYTE". The startAddr here will be forced to align to CACHE64_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.

- size_byte – size of the memory to be cleaned, should be larger than 0.

void CACHE64_CleanInvalidateCache(CACHE64_CTRL_Type *base)

Cleans and invalidates the cache.

Parameters

- base – CACHE64_CTRL peripheral base address.

void CACHE64_CleanInvalidateCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and invalidate cache by range.

Note: Address and size should be aligned to "CACHE64_LINESIZE_BYTE". The startAddr here will be forced to align to CACHE64_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0.

static inline void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates instruction cache by range.

Note: Address and size should be aligned to CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated, should be larger than 0.

static inline void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates data cache by range.

Note: Address and size should be aligned to CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated, should be larger than 0.

static inline void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Clean data cache by range.

Note: Address and size should be aligned to CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be cleaned, should be larger than 0.

static inline void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates data cache by range.

Note: Address and size should be aligned to CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0.

FSL_CACHE_DRIVER_VERSION

cache driver version.

enum _cache64_policy

Level 2 cache controller way size.

Values:

enumerator kCACHE64_PolicyNonCacheable

Non-cacheable

enumerator kCACHE64_PolicyWriteThrough

Write through

enumerator kCACHE64_PolicyWriteBack

Write back

typedef enum *_cache64_policy* cache64_policy_t

Level 2 cache controller way size.

typedef struct *_cache64_config* cache64_config_t

CACHE64 configuration structure.

CACHE64_LINESIZE_BYTE
cache line size.

CACHE64_REGION_NUM
cache region number.

CACHE64_REGION_ALIGNMENT
cache region alignment.

struct _cache64_config
#include <fsl_cache.h> CACHE64 configuration structure.

Public Members

uint32_t boundaryAddr[(3U) - 1]
< The cache controller can divide whole memory into 3 regions. Boundary address is the FlexSPI internal address (start from 0) instead of system address (start from FlexSPI AMBA base) to split adjacent regions and must be 1KB aligned. The boundary address itself locates in upper region. Cacheable policy for each region.

2.3 CDOG

status_t CDOG_Init(CDOG_Type *base, *cdog_config_t* *conf)
Initialize CDOG.

This function initializes CDOG block and setting.

Parameters

- base – CDOG peripheral base address
- conf – CDOG configuration structure

Returns
Status of the init operation

void CDOG_Deinit(CDOG_Type *base)
Deinitialize CDOG.

This function deinitializes CDOG secure counter.

Parameters

- base – CDOG peripheral base address

void CDOG_GetDefaultConfig(*cdog_config_t* *conf)
Sets the default configuration of CDOG.

This function initialize CDOG config structure to default values.

Parameters

- conf – CDOG configuration structure

void CDOG_Stop(CDOG_Type *base, uint32_t stop)

Stops secure counter and instruction timer.

This function stops instruction timer and secure counter. This also change state od CDOG to IDLE.

Parameters

- base – CDOG peripheral base address
- stop – expected value which will be compared with value of secure counter

void CDOG_Start(CDOG_Type *base, uint32_t reload, uint32_t start)

Sets secure counter and instruction timer values.

This function sets value in RELOAD and START registers for instruction timer and secure counter

Parameters

- base – CDOG peripheral base address
- reload – reload value
- start – start value

void CDOG_Check(CDOG_Type *base, uint32_t check)

Checks secure counter.

This function compares stop value in handler with secure counter value by writting to RELOAD refister.

Parameters

- base – CDOG peripheral base address
- check – expected (stop) value

void CDOG_Set(CDOG_Type *base, uint32_t stop, uint32_t reload, uint32_t start)

Sets secure counter and instruction timer values.

This function sets value in STOP, RELOAD and START registers for instruction timer and secure counter.

Parameters

- base – CDOG peripheral base address
- stop – expected value which will be compared with value of secure counter
- reload – reload value for instruction timer
- start – start value for secure timer

void CDOG_Add(CDOG_Type *base, uint32_t add)

Add value to secure counter.

This function add specified value to secure counter.

Parameters

- base – CDOG peripheral base address.
- add – Value to be added.

void CDOG_Add1(CDOG_Type *base)

Add 1 to secure counter.

This function add 1 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Add16(CDOG_Type *base)

Add 16 to secure counter.

This function add 16 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Add256(CDOG_Type *base)

Add 256 to secure counter.

This function add 256 to secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub(CDOG_Type *base, uint32_t sub)

brief Substract value to secure counter

This function substract specified value to secure counter.

param base CDOG peripheral base address. param sub Value to be substracted.

void CDOG_Sub1(CDOG_Type *base)

Substract 1 from secure counter.

This function substract specified 1 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub16(CDOG_Type *base)

Substract 16 from secure counter.

This function substract specified 16 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_Sub256(CDOG_Type *base)

Substract 256 from secure counter.

This function substract specified 256 from secure counter.

Parameters

- base – CDOG peripheral base address.

void CDOG_WritePersistent(CDOG_Type *base, uint32_t value)

Set the CDOG persistent word.

Parameters

- base – CDOG peripheral base address.
- value – The value to be written.

uint32_t CDOG_ReadPersistent(CDOG_Type *base)

Get the CDOG persistent word.

Parameters

- base – CDOG peripheral base address.

Returns

The persistent word.

FSL_CDOG_DRIVER_VERSION

Defines CDOG driver version 2.1.3.

Change log:

- Version 2.1.3
  - Re-design multiple instance IRQs and Clocks
  - Add fix for RESTART command errata
- Version 2.1.2
  - Support multiple IRQs
  - Fix default CONTROL values
- Version 2.1.1
  - Remove bit CONTROL[CONTROL_CTRL]
- Version 2.1.0
  - Rename CWT to CDOG
- Version 2.0.2
  - Fix MISRA-2012 issues
- Version 2.0.1
  - Fix doxygen issues
- Version 2.0.0
  - initial version

enum ___cdog_debug_Action_ctrl_enum

Values:

enumerator kCDOG_DebugHaltCtrl_Run

enumerator kCDOG_DebugHaltCtrl_Pause

enum ___cdog_irq_pause_ctrl_enum

Values:

enumerator kCDOG_IrqPauseCtrl_Run

enumerator kCDOG_IrqPauseCtrl_Pause

enum ___cdog_fault_ctrl_enum

Values:

enumerator kCDOG_FaultCtrl_EnableReset

enumerator kCDOG_FaultCtrl_EnableInterrupt

enumerator kCDOG_FaultCtrl_NoAction

enum ___code_lock_ctrl_enum

Values:

enumerator kCDOG_LockCtrl_Lock

enumerator kCDOG_LockCtrl_Unlock

typedef uint32_t secure_counter_t

SC_ADD(add)

SC_ADD1

SC_ADD16

SC_ADD256

SC_SUB(sub)

SC_SUB1

SC_SUB16

SC_SUB256

SC_CHECK(val)

struct cdog_config_t

#include <fsl_cdog.h>

2.4 Clock Driver

enum _clock_usb_src

USB clock source definition.

Values:

enumerator kCLOCK_Usb480M

Use 480M.

enumerator kCLOCK_UsbSrcUnused

Used when the function does not care the clock source.

enum _clock_usb_phy_src

Source of the USB HS PHY.

Values:

enumerator kCLOCK_Usbphy480M
Use 480M.

enum _clock_ip_name
Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.
Values:

enumerator kCLOCK_IpInvalid
Invalid Ip Name.

enumerator kCLOCK_Xcache1
Clock gate name: Code cache

enumerator kCLOCK_Xcache0
Clock gate name: System cache

enumerator kCLOCK_Ocotp0
Clock gate name: VDD2 OTP0

enumerator kCLOCK_Sleepcon0
Clock gate name: SLEEPCON_CMPT

enumerator kCLOCK_Syscon0
Clock gate name: SYSCON_CMPT

enumerator kCLOCK_Glikey0
Clock gate name: GLIKEY0

enumerator kCLOCK_Glikey3
Clock gate name: GLIKEY3

enumerator kCLOCK_TpiuTraceClkin
Clock gate name: TPIU_TRACECLKIN

enumerator kCLOCK_SWOTraceClkin
Clock gate name: SWO_TRACECLKIN

enumerator kCLOCK_Tsclk
Clock gate name: TRACE

enumerator kCLOCK_Dma0
Clock gate name: DMA0

enumerator kCLOCK_Dma1
Clock gate name: DMA1

enumerator kCLOCK_PkcRam
Clock gate name: PKC RAM

enumerator kCLOCK_Pkc
Clock gate name: PKC

enumerator kCLOCK_Romcp
Clock gate name: ROMCP

enumerator kCLOCK_Xspi0
Clock gate name: XSPI0

enumerator kCLOCK_Xspi1
Clock gate name: XSPI1

enumerator kCLOCK_Cache64ctrl0
Clock gate name: CACHE64_0

enumerator kCLOCK_Cache64ctrl1
Clock gate name: CACHE64_1

enumerator kCLOCK_Puf
Clock gate name: QK_SUBSYS

enumerator kCLOCK_Mmu0
Clock gate name: MMU0

enumerator kCLOCK_Mmu1
Clock gate name: MMU1

enumerator kCLOCK_Gpio0
Clock gate name: GPIO0

enumerator kCLOCK_Gpio1
Clock gate name: GPIO1

enumerator kCLOCK_Gpio2
Clock gate name: GPIO2

enumerator kCLOCK_Gpio3
Clock gate name: GPIO3

enumerator kCLOCK_Gpio4
Clock gate name: GPIO4

enumerator kCLOCK_Gpio5
Clock gate name: GPIO5

enumerator kCLOCK_Gpio6
Clock gate name: GPIO6

enumerator kCLOCK_Gpio7
Clock gate name: GPIO7

enumerator kCLOCK_Sct0
Clock gate name: SCT0

enumerator kCLOCK_Cdog0
Clock gate name: CDOG0

enumerator kCLOCK_Cdog1
Clock gate name: CDOG1

enumerator kCLOCK_Cdog2
Clock gate name: CDOG2

enumerator kCLOCK_LPFlexComm0
Clock gate name: LP_Flexcomm0

enumerator kCLOCK_LPFlexComm1
Clock gate name: LP_Flexcomm1

enumerator kCLOCK_LPUart0
Clock gate name: LPUART0

enumerator kCLOCK_LPUart1
Clock gate name: LPUART1

enumerator kCLOCK_LPI2c0
Clock gate name: LPI2C0

enumerator kCLOCK_LPI2c1
Clock gate name: LPI2C1

enumerator kCLOCK_LPSpi0
Clock gate name: LPSPI0

enumerator kCLOCK_LPSpi1
Clock gate name: LPSPI1

enumerator kCLOCK_LPFlexComm2
Clock gate name: LP_Flexcomm2

enumerator kCLOCK_LPFlexComm3
Clock gate name: LP_Flexcomm3

enumerator kCLOCK_LPFlexComm4
Clock gate name: LP_Flexcomm4

enumerator kCLOCK_LPFlexComm5
Clock gate name: LP_Flexcomm5

enumerator kCLOCK_LPFlexComm6
Clock gate name: LP_Flexcomm6

enumerator kCLOCK_LPFlexComm7
Clock gate name: LP_Flexcomm7

enumerator kCLOCK_LPFlexComm8
Clock gate name: LP_Flexcomm8

enumerator kCLOCK_LPFlexComm9
Clock gate name: LP_Flexcomm9

enumerator kCLOCK_LPFlexComm10
Clock gate name: LP_Flexcomm10

enumerator kCLOCK_LPFlexComm11
Clock gate name: LP_Flexcomm11

enumerator kCLOCK_LPFlexComm12
Clock gate name: LP_Flexcomm12

enumerator kCLOCK_LPFlexComm13
Clock gate name: LP_Flexcomm13

enumerator kCLOCK_LPUart2
Clock gate name: LPUART2

enumerator kCLOCK_LPUart3
Clock gate name: LPUART3

enumerator kCLOCK_LPUart4
Clock gate name: LPUART4

enumerator kCLOCK_LPUart5
Clock gate name: LPUART5

enumerator kCLOCK_LPUart6
Clock gate name: LPUART6

enumerator kCLOCK_LPUart7
Clock gate name: LPUART7

enumerator kCLOCK_LPUart8
Clock gate name: LPUART8

enumerator kCLOCK_LPUart9
Clock gate name: LPUART9

enumerator kCLOCK_LPUart10
Clock gate name: LPUART10

enumerator kCLOCK_LPUart11
Clock gate name: LPUART11

enumerator kCLOCK_LPUart12
Clock gate name: LPUART12

enumerator kCLOCK_LPUart13
Clock gate name: LPUART13

enumerator kCLOCK_LPI2c2
Clock gate name: LPI2C2

enumerator kCLOCK_LPI2c3
Clock gate name: LPI2C3

enumerator kCLOCK_LPI2c4
Clock gate name: LPI2C4

enumerator kCLOCK_LPI2c5
Clock gate name: LPI2C5

enumerator kCLOCK_LPI2c6
Clock gate name: LPI2C6

enumerator kCLOCK_LPI2c7
Clock gate name: LPI2C7

enumerator kCLOCK_LPI2c8
Clock gate name: LPI2C8

enumerator kCLOCK_LPI2c9
Clock gate name: LPI2C9

enumerator kCLOCK_LPI2c10
Clock gate name: LPI2C10

enumerator kCLOCK_LPI2c11
Clock gate name: LPI2C11

enumerator kCLOCK_LPI2c12
Clock gate name: LPI2C12

enumerator kCLOCK_LPI2c13
Clock gate name: LPI2C13

enumerator kCLOCK_LPSpi2
Clock gate name: LPSPI2

enumerator kCLOCK_LPSpi3
Clock gate name: LPSPI3

enumerator kCLOCK_LPSpi4
Clock gate name: LPSPI4

enumerator kCLOCK_LPSpi5
Clock gate name: LPSPI5

enumerator kCLOCK_LPSpi6
Clock gate name: LPSPI6

enumerator kCLOCK_LPSpi7
Clock gate name: LPSPI7

enumerator kCLOCK_LPSpi8
Clock gate name: LPSPI8

enumerator kCLOCK_LPSpi9
Clock gate name: LPSPI9

enumerator kCLOCK_LPSpi10
Clock gate name: LPSPI10

enumerator kCLOCK_LPSpi11
Clock gate name: LPSPI11

enumerator kCLOCK_LPSpi12
Clock gate name: LPSPI12

enumerator kCLOCK_LPSpi13
Clock gate name: LPSPI13

enumerator kCLOCK_Sai0
Clock gate name: SAI0

enumerator kCLOCK_Sai1
Clock gate name: SAI1

enumerator kCLOCK_Sai2
Clock gate name: SAI2

enumerator kCLOCK_I3c0
Clock gate name: I3C0

enumerator kCLOCK_I3c1
Clock gate name: I3C1

enumerator kCLOCK_Crc0
Clock gate name: CRC0

enumerator kCLOCK_Wwdt0
Clock gate name: WWDT0

enumerator kCLOCK_Wwdt1
Clock gate name: WWDT1

enumerator kCLOCK_Ct32b0
Clock gate name: CTIMER0

enumerator kCLOCK_Ct32b1
Clock gate name: CTIMER1

enumerator kCLOCK_Ct32b2
Clock gate name: CTIMER2

enumerator kCLOCK_Ct32b3
Clock gate name: CTIMER3

enumerator kCLOCK_Ct32b4
Clock gate name: CTIMER4

enumerator kCLOCK_Mrt0
Clock gate name: Mrt0

enumerator kCLOCK_Utick0
Clock gate name: Utick0

enumerator kCLOCK_Sema424
Clock gate name: SEMA42_4

enumerator kCLOCK_Mu4
Clock gate name: MU4

enumerator kCLOCK_Pint0
Clock gate name: PINT0

enumerator kCLOCK_Pvts0
Clock gate name: PVTS0

enumerator kCLOCK_InputMux0
Clock gate name: PMUX_CMPT_SPLITTER

enumerator kCLOCK_Freqme0
Clock gate name: FREQME0

enumerator kCLOCK_SafoSgi
Clock gate name: SAFO_SGI

enumerator kCLOCK_Trace
Clock gate name: TRACE

enumerator kCLOCK_Prince0
Clock gate name: PRINCE0

enumerator kCLOCK_Prince1
Clock gate name: PRINCE1

enumerator kCLOCK_PrinceExe
Clock gate name: PRINCE_EXE

enumerator kCLOCK_Syspm0
Clock gate name: CMX_PERFMON0

enumerator kCLOCK_Syspm1
Clock gate name: CMX_PERFMON1

enumerator kCLOCK_Hifi4
Clock gate name: HIFI4

enumerator kCLOCK_Npu0
Clock gate name: NPU0

enumerator kCLOCK_CompAccessRamArbiter1
Clock gate name: COMP_ACCESS_RAM_ARBITER1

enumerator kCLOCK_Iopctl0
Clock gate name: IOMUXC_VDD2

enumerator kCLOCK_Hifi4AccessRamArbiter1
Clock gate name: HIFI4_ACCESS_RAM_ARBITER1

enumerator kCLOCK_MediaAccessRamArbiter0
Clock gate name: MEDIA_ACCESS_RAM_ARBITER0

enumerator kCLOCK_Sleepcon1
Clock gate name: SLEEPCONCPU1

enumerator kCLOCK_Syscon1
Clock gate name: SYSCONSENSE1

enumerator kCLOCK_SenseAccessRamArbiter0
Clock gate name: SENSE_ACCESS_RAM_ARBITER0

enumerator kCLOCK_Hifi1
Clock gate name: HIFI1

enumerator kCLOCK_Dma2
Clock gate name: DMA2

enumerator kCLOCK_Dma3
Clock gate name: DMA3

enumerator kCLOCK_LPFlexComm17
Clock gate name: LP_Flexcomm17

enumerator kCLOCK_LPFlexComm18
Clock gate name: LP_Flexcomm18

enumerator kCLOCK_LPFlexComm19
Clock gate name: LP_Flexcomm19

enumerator kCLOCK_LPFlexComm20
Clock gate name: LP_Flexcomm20

enumerator kCLOCK_LPI2c17
Clock gate name: LPI2C17

enumerator kCLOCK_LPI2c18
Clock gate name: LPI2C18

enumerator kCLOCK_LPI2c19
Clock gate name: LPI2C19

enumerator kCLOCK_LPI2c20
Clock gate name: LPI2C20

enumerator kCLOCK_LPSpi17
Clock gate name: LPSPI17

enumerator kCLOCK_LPSpi18
Clock gate name: LPSPI18

enumerator kCLOCK_LPSpi19
Clock gate name: LPSPI19

enumerator kCLOCK_LPSpi20
Clock gate name: LPSPI20

enumerator kCLOCK_LPUart17
Clock gate name: LPUART17

enumerator kCLOCK_LPUart18
Clock gate name: LPUART18

enumerator kCLOCK_LPUart19
Clock gate name: LPUART19

enumerator kCLOCK_LPUart20
Clock gate name: LPUART20

enumerator kCLOCK_Sai3
Clock gate name: SAI3

enumerator kCLOCK_I3c2
Clock gate name: I3C2

enumerator kCLOCK_I3c3
Clock gate name: I3C3

enumerator kCLOCK_Gpio8
Clock gate name: GPIO8

enumerator kCLOCK_Gpio9
Clock gate name: GPIO9

enumerator kCLOCK_Gpio10
Clock gate name: GPIO10

enumerator kCLOCK_Pint1
Clock gate name: PINT1

enumerator kCLOCK_Ct32b5
Clock gate name: CTIMER5

enumerator kCLOCK_Ct32b6
Clock gate name: CTIMER6

enumerator kCLOCK_Ct32b7
Clock gate name: CTIMER7

enumerator kCLOCK_Mrt1
Clock gate name: Mrt1

enumerator kCLOCK_Utick1
Clock gate name: Utick1

enumerator kCLOCK_Cdog3
Clock gate name: CDOG3

enumerator kCLOCK_Cdog4
Clock gate name: CDOG4

enumerator kCLOCK_Mu3
Clock gate name: MU3

enumerator kCLOCK_Sema423
Clock gate name: SEMA42_3

enumerator kCLOCK_Wwdt2
Clock gate name: WWDT2

enumerator kCLOCK_Wwdt3
Clock gate name: WWDT3

enumerator kCLOCK_Pvts1
Clock gate name: PINT1

enumerator kCLOCK_InputMux1
Clock gate name: PMUX_SNS_SPLITTER

enumerator kCLOCK_Syscon2
Clock gate name: SYSCON_COMM

enumerator kCLOCK_Iopctl2
Clock gate name: IOMUXC_VDDN

enumerator kCLOCK_Cpu1
Clock gate name: CPU1

enumerator kCLOCK_Mu0
Clock gate name: MU0

enumerator kCLOCK_Mu1
Clock gate name: MU1

enumerator kCLOCK_Mu2
Clock gate name: MU2

enumerator kCLOCK_OsTimer
Clock gate name: OsTimer

enumerator kCLOCK_Sema420
Clock gate name: SEMA42_0

enumerator kCLOCK_Sdadc0
Clock gate name: SDADC0

enumerator kCLOCK_Adc0
Clock gate name: SARADC0

enumerator kCLOCK_Acmp0
Clock gate name: Acmp0

enumerator kCLOCK_Pdm
Clock gate name: MICFIL(PDM)

enumerator kCLOCK_Glikey4
Clock gate name: GLIKEY_SYSCON1

enumerator kCLOCK_Dbg
Clock gate name: DBG

enumerator kCLOCK_Syscon3
Clock gate name: SYSCON_SENSE0

enumerator kCLOCK_Iopctl1
Clock gate name: IOMUXC_VDD1

enumerator kCLOCK_Glikey1
Clock gate name: GLIKEY1

enumerator kCLOCK_LPI2c15
Clock gate name: LPI2C

enumerator kCLOCK_MediaAccessRamArbiter1
Clock gate name: MEDIA_ACCESS_RAM_ARBITER1

enumerator kCLOCK_Axi0
Clock gate name: AXI0

enumerator kCLOCK_Gpu
Clock gate name: VGPU

enumerator kCLOCK_MipiDsiCtrl
Clock gate name: MIPIDSI

enumerator kCLOCK_LPSpi16
Clock gate name: LPSPI16

enumerator kCLOCK_LPSpi14
Clock gate name: LPSPI14

enumerator kCLOCK_Xspi2
Clock gate name: XSPI2

enumerator kCLOCK_Mmu2
Clock gate name: MMU2

enumerator kCLOCK_Glikey5
Clock gate name: GLIKEY_SYSCON2

enumerator kCLOCK_Flexio
Clock gate name: FLEXIO0

enumerator kCLOCK_Lcdif
Clock gate name: LCDIF(DCN)

enumerator kCLOCK_Syscon4
Clock gate name: SYSCONMEDIA

enumerator kCLOCK_JpgDecoder
Clock gate name: JPG_DECODER

enumerator kCLOCK_PngDecoder
Clock gate name: PNG_DECODER

enumerator kCLOCK_Ezhv
Clock gate name: EZHV

enumerator kCLOCK_AxbsEzh
Clock gate name: AXBS_EZH

enumerator kCLOCK_Glikey2
Clock gate name: GLIKEY2

enumerator kCLOCK_Usb0
Clock gate name: USB0

enumerator kCLOCK_Usb1
Clock gate name: USB1

enumerator kCLOCK_Usdhc0
Clock gate name: USHDC0

enumerator kCLOCK_Usdhc1
Clock gate name: USDHC1

enumerator kCLOCK_Gdet3Ref
Clock gate name: GDET3 Reference clock

enumerator kCLOCK_Rtc
RTC functional clock gating.

enumerator kCLOCK_Gdet3
GDET3 functional clock gating.

enumerator kCLOCK_UsbphyRef
USBPHY referrence clock gating.

enum _clock_name
Clock name used to get clock frequency.
Values:

enumerator kCLOCK_CoreSysClk
Core clock (aka system clock)

enumerator kCLOCK_BusClk
Bus clock (AHB/APB clock, aka HCLK)

enumerator kCLOCK_MclkClk
MCLK, to MCLK pin

enumerator kCLOCK_Wdt2Clk
Watchdog1

enumerator kCLOCK_Wdt3Clk
Watchdog1

enumerator kCLOCK_Hifi1CpuClk
Hifi1 DSP clock

enumerator kCLOCK_LPFlexComm17Clk
Flexcomm17Clock

enumerator kCLOCK_LPFlexComm18Clk
Flexcomm18Clock

enumerator kCLOCK_LPFlexComm19Clk
Flexcomm19Clock

enumerator kCLOCK_LPFlexComm20Clk
Flexcomm20Clock

enumerator kCLOCK_Vdd1ClockOutClk
VDD1 CLOCKOUT

enumerator kCLOCK_AdcClk
ADC

enumerator kCLOCK_Xspi2Clk
XSPI1

enumerator kCLOCK_SctClk
SCT

enumerator kCLOCK_SystickClk
Systick

enumerator kCLOCK_Sdio0Clk
SDIO0

enumerator kCLOCK_Sdio1Clk
SDIO1

enumerator kCLOCK_I3cClk
I3C

enumerator kCLOCK_Usb0Clk
USB0

enumerator kCLOCK_Usb1Clk
USB1

enumerator kCLOCK_PdmClk
Digital Mic clock

enumerator kCLOCK_AcmpClk
Acmp clock

enumerator kCLOCK_FlexioClk
FlexIO

enumerator kCLOCK_LPSpi14Clk
LPSPI14

enumerator kCLOCK_LPI2c15Clk
LPI2C15

enumerator kCLOCK_LPSpi16Clk
LPSPI16

enumerator kCLOCK_VgpuClk
VGPU Core

enumerator kCLOCK_LcdifClk
LCDIF Clock

enumerator kCLOCK_MipiDphyClk
MIPI D-PHY Bit Clock

enumerator kCLOCK_MipiDphyEscRxClk
MIPI D-PHY RX Clock

enumerator kCLOCK_MipiDphyEscTxClk
MIPI D-PHY TX Clock

enum _clock_attach_id
The enumerator of clock attach Id for multi sources clock slices. Not all Muxer has MUX output enable control, if no mux output enable control, the mux output will be always enabled.

Values:

enumerator kFRO1_DIV3_to_SENSE_BASE
Attach FRO1 divided-by-3 clock to Sense Base Clock.

enumerator kFRO1_DIV1_to_SENSE_BASE
Attach FRO1 max clock to Sense Base Clock.

enumerator kFRO2_DIV3_to_SENSE_BASE
Attach FRO2 divided-by-3 clock to Sense Base Clock.

enumerator kLPOSC_to_SENSE_BASE
Attach LPOSC to Sense Base Clock.

enumerator kSENSE_BASE_to_SENSE_DSP
Attach Sense base clock to Sense DSP Clock.

enumerator kFRO2_DIV1_to_SENSE_DSP
Attach FRO2 max clock to Sense DSP Clock.

enumerator kAUDIO_PLL_PFD1_to_SENSE_DSP
Attach Audio PLL PFD1 clock to Sense DSP Clock.

enumerator kFRO1_DIV1_to_SENSE_DSP
Attach FRO1 max clock to Sense DSP Clock.

enumerator kNONE_to_SENSE_DSP
Attach NONE to Sense DSP Clock.

enumerator kSENSE_BASE_to_SAI3
Attach Sense base clock to SAI3 Clock.

enumerator kFRO2_DIV8_to_SAI3
Attach FRO2_DIV8 clock to SAI3 Clock.

enumerator kFRO2_DIV1_to_SAI3
Attach FRO2 clock to SAI3 Clock.

enumerator kAUDIO_VDD1_to_SAI3
Attach audio_clk to SAI3 Clock.

enumerator kNONE_to_SAI3
Attach NONE to SAI3 Clock.

enumerator kSENSE_BASE_to_UTICK
Attach Sense base clock to UTICK Functional Clock.

enumerator kLPOSC_to_UTICK
Attach LPOSC clock to UTICK Functional Clock.

enumerator kFRO2_DIV1_to_UTICK
Attach FRO2 max clock to UTICK Functional Clock.

enumerator kFRO1_DIV2_to_UTICK
Attach FRO1_DIV2 clock to UTICK Functional Clock.

enumerator kNONE_to_UTICK
Attach NONE to UTICK Functional Clock.

enumerator kLPOSC_to_WWDT2
Attach LPOSC clock to WWDT2 Functional Clock.

enumerator kNONE_to_WWDT2
Attach NONE to WWDT2 Functional Clock.

enumerator kLPOSC_to_WWDT3
Attach LPOSC clock to WWDT3 Functional Clock.

enumerator kNONE_to_WWDT3
Attach NONE to WWDT3 Functional Clock.

enumerator kSENSE_BASE_to_SYSTICK
Attach Sense base clock to SYSTICK Functional Clock.

enumerator kLPOSC_to_SYSTICK
Attach LPOSC clock to SYSTICK Functional Clock.

enumerator k32KHZ_WAKE_to_SYSTICK
Attach 32 kHz wake clock(switch to other clock source before sense vdd1 enter SRPG mode.) to SYSTICK Functional Clock.

enumerator kOSC_CLK_to_SYSTICK
Attach OSC clock to SYSTICK Functional Clock.

enumerator kNONE_to_SYSTICK
Attach NONE to SYSTICK Functional Clock.

enumerator kSENSE_BASE_to_CTIMER5
Attach Sense base clock to CTIMER5 Clock.

enumerator kAUDIO_VDD1_to_CTIMER5
Attach audio_clk to CTIMER5 Clock.

enumerator kFRO2_DIV1_to_CTIMER5
Attach FRO2 Max clock to CTIMER5 Clock.

enumerator k32KHZ_WAKE_to_CTIMER5
Attach 32KHz wake clock to CTIMER5 Clock.

enumerator kNONE_to_CTIMER5
Attach NONE to CTIMER5 Clock.

enumerator kSENSE_BASE_to_CTIMER6
Attach Sense base clock to CTIMER6 Clock.

enumerator kAUDIO_VDD1_to_CTIMER6
Attach audio_clk to CTIMER6 Clock.

enumerator kFRO2_DIV1_to_CTIMER6
Attach FRO2 Max clock to CTIMER6 Clock.

enumerator k32KHZ_WAKE_to_CTIMER6
Attach 32KHz wake clock to CTIMER6 Clock.

enumerator kNONE_to_CTIMER6
Attach NONE to CTIMER6 Clock.

enumerator kSENSE_BASE_to_CTIMER7
Attach Sense base clock to CTIMER7 Clock.

enumerator kAUDIO_VDD1_to_CTIMER7
Attach audio_clk to CTIMER7 Clock.

enumerator kFRO2_DIV1_to_CTIMER7
Attach FRO2 Max clock to CTIMER7 Clock.

enumerator k32KHZ_WAKE_to_CTIMER7
Attach 32KHz wake clock to CTIMER7 Clock.

enumerator kNONE_to_CTIMER7
Attach NONE to CTIMER7 Clock.

enumerator kSENSE_BASE_to_I3C23
Attach Sense base clock to I3C2 and I3C3 Functional Clock.

enumerator kFRO2_DIV1_to_I3C23
Attach FRO2 max clock to I3C2 and I3C3 Functional Clock.

enumerator kFRO1_DIV8_to_I3C23
Attach FRO2_DIV8 clock to I3C2 and I3C3 Functional Clock.

enumerator kOSC_CLK_to_I3C23
Attach OSC clock to I3C2 and I3C3 Functional Clock.

enumerator kNONE_to_I3C23
Attach NONE to I3C2 and I3C3 Functional Clock.

enumerator kMCLK_to_AUDIO_VDD1
Attach MCLK IN from Pad to Audio VDD1 Clock.

enumerator kOSC_to_AUDIO_VDD1
Attach OSC clock to Audio VDD1 Clock.

enumerator kFRO2_DIV8_to_AUDIO_VDD1
Attach FRO2_DIV8 clock to Audio VDD1 Clock.

enumerator kAUDIO_PLL_PFD3_to_AUDIO_VDD1
Attach AUDIO PLL PFD3 clock to Audio VDD1 Clock.

enumerator kSENSE_BASE_to_FLEXCOMM17
Attach Sense base clock to FLEXCOMM17.

enumerator kFRO2_DIV1_to_FLEXCOMM17
Attach FRO2 max clock to FLEXCOMM17.

enumerator kFRO1_DIV1_to_FLEXCOMM17
Attach FRO1 max clock to FLEXCOMM17.

enumerator k32KHZ_WAKE_to_FLEXCOMM17
Attach 32k_wake_clk to FLEXCOMM17.

enumerator kNONE_to_FLEXCOMM17
Attach NONE to FLEXCOMM17.

enumerator kSENSE_BASE_to_FLEXCOMM18
Attach Sense base clock to FLEXCOMM18.

enumerator kFRO2_DIV1_to_FLEXCOMM18
Attach FRO2 max clock to FLEXCOMM18.

enumerator kFRO1_DIV1_to_FLEXCOMM18
Attach FRO1 max clock to FLEXCOMM18.

enumerator k32KHZ_WAKE_to_FLEXCOMM18
Attach 32k_wake_clk to FLEXCOMM18.

enumerator kNONE_to_FLEXCOMM18
Attach NONE to FLEXCOMM18.

enumerator kSENSE_BASE_to_FLEXCOMM19
Attach Sense base clock to FLEXCOMM19.

enumerator kFRO2_DIV1_to_FLEXCOMM19
Attach FRO2 max clock to FLEXCOMM19.

enumerator kFRO1_DIV1_to_FLEXCOMM19
Attach FRO1 max clock to FLEXCOMM19.

enumerator k32KHZ_WAKE_to_FLEXCOMM19
Attach 32k_wake_clk to FLEXCOMM19.

enumerator kNONE_to_FLEXCOMM19
Attach NONE to FLEXCOMM19.

enumerator kSENSE_BASE_to_FLEXCOMM20
Attach Sense base clock to FLEXCOMM20.

enumerator kFRO2_DIV1_to_FLEXCOMM20
Attach FRO2 max clock to FLEXCOMM20.

enumerator kFRO1_DIV1_to_FLEXCOMM20
Attach FRO1 max clock to FLEXCOMM20.

enumerator k32KHZ_WAKE_to_FLEXCOMM20
Attach 32k_wake_clk to FLEXCOMM20.

enumerator kNONE_to_FLEXCOMM20
Attach NONE to FLEXCOMM20.

enumerator kCOMMON_BASE_to_COMMON_VDDN
Attach Common base clock to Common VDDN Clock.

enumerator kMAIN_PLL_PFD3_to_COMMON_VDDN
Attach Main PLL PFD3 clock to Common VDDN Clock.

enumerator kFRO1_DIV1_to_COMMON_VDDN
Attach FRO1 max clock to Common VDDN Clock.

enumerator kOSC_CLK_to_COMMON_VDDN
Attach OSC clock to Common VDDN Clock.

enumerator kOSC_CLK_to_USB_24MHZ
Attach OSC_CLK clock to 24MHz USB OSC Clock.

enumerator kFRO1_DIV8_to_USB_24MHZ
Attach FRO1_DIV8 clock to 24MHz USB OSC Clock.

enumerator kFRO1_DIV3_to_COMMON_BASE
Attach Fro1 divided-by-3 to Common Base Clock.

enumerator kFRO1_DIV1_to_COMMON_BASE
Attach Fro1 divided-by-1 to Common Base Clock.

enumerator kFRO0_DIV3_to_COMMON_BASE
Attach Fro0 divided-by-3 to Common Base Clock.

enumerator kLPOSC_to_COMMON_BASE
Attach LPOSC to Common Base Clock.

enumerator kOSC_CLK_to_EUSB_24MHZ
Attach OSC_CLK clock to 24MHz eUSB OSC Clock.

enumerator kFRO1_DIV8_to_EUSB_24MHZ
Attach FRO1_DIV8 clock to 24MHz eUSB OSC Clock.

enumerator kFRO1_DIV8_to_MAIN_PLL0
Attach FRO1_DIV8 clock to Main PLL0 Clock.

enumerator kOSC_CLK_to_MAIN_PLL0
Attach OSC_CLK clock to Main PLL0 Clock.

enumerator kFRO1_DIV8_to_AUDIO_PLL0
Attach FRO1_DIV8 to Audio PLL0 Clock.

enumerator kOSC_CLK_to_AUDIO_PLL0
Attach OSC_CLK clock (User-Selectable) to Audio PLL0 Clock.

enumerator kSENSE_BASE_to_SENSE_MAIN
Attach Sense base clock to Sense main clock.

enumerator kFRO2_DIV1_to_SENSE_MAIN
Attach FRO2 max clock to Sense main clock.

enumerator kAUDIO_PLL_PFD3_to_SENSE_MAIN
Attach Audio PLL PFD3 clock to Sense main clock.

enumerator kFRO1_DIV1_to_SENSE_MAIN
Attach FRO1 max clock to Sense main clock.

enumerator kSENSE_BASE_to_SENSE_RAM
Attach Sense base clock to Sense RAM clock.

enumerator kFRO2_DIV1_to_SENSE_RAM
Attach FRO2 max clock to Sense RAM clock.

enumerator kAUDIO_PLL_PFD2_to_SENSE_RAM
Attach Audio PLL PFD2 clock to Sense RAM clock.

enumerator kFRO1_DIV1_to_SENSE_RAM
Attach FRO1 max clock to Sense RAM clock.

enumerator kSENSE_BASE_to_OSTIMER
Attach Sense Base to OSTIMER.

enumerator k32KHZ_WAKE_to_OSTIMER
Attach 32k_wake_clk to OSTIMER.

enumerator kLPOSC_to_OSTIMER
Attach LPOSC to OSTIMER.

enumerator kOSC_CLK_to_OSTIMER
Attach OSC clock to OSTIMER.

enumerator kNONE_to_OSTIMER
Attach NONE to OSTIMER.

enumerator kSENSE_BASE_to_SDADC
Attach Sense base clock to SDADC Functional Clock.

enumerator kFRO2_DIV8_to_SDADC
Attach FRO2_DIV8 clock to SDADC Functional Clock.

enumerator kAUDIO_PLL_to_SDADC
Attach Audio PLL VCO clock to SDADC Functional Clock.

enumerator kAUDIO_VDD1_to_SDADC
Attach audio_clk to SDADC Functional Clock.

enumerator kNONE_to_SDADC
Attach NONE to SDADC Functional Clock.

enumerator kSENSE_BASE_to_ADC
Attach Sense base clock to SARADC Functional Clock.

enumerator kMAIN_PLL_PFD1_to_ADC
Attach main_pll_pfd1 to SARADC Functional Clock.

enumerator kFRO2_DIV1_to_ADC
Attach FRO0 max clock to SARADC Functional Clock.

enumerator kOSC_CLK_to_ADC
Attach osc_clk to SARADC Functional Clock.

enumerator kNONE_to_ADC
Attach NONE to SARADC Functional Clock.

enumerator kOSC32K_to_32K_WAKE
Attach OSC 32K to Wake 32k Clock.

enumerator kLPOSC_DIV32_to_32K_WAKE
Attach LPOSC clock divided by 32 by default to Wake 32k Clock.

enumerator kNONE_to_32K_WAKE
Attach NONE to Wake 32k Clock.

enumerator kSENSE_BASE_to_MICFIL0
Attach Sense base clock to MICFIL0 Functional Clock.

enumerator kAUDIO_PLL_PFD3_to_MICFIL0
Attach Audio PLL PFD3 clock to MICFIL0 Functional Clock.

enumerator kFRO2_DIV1_to_MICFIL0
Attach FRO2 MAX clock to MICFIL0 Functional Clock.

enumerator kAUDIO_VDD1_to_MICFIL0
Attach Audio clock to MICFIL0 Functional Clock.

enumerator kNONE_to_MICFIL0
Attach NONE to MICFIL0 Functional Clock.

enumerator kSENSE_BASE_to_LPI2C15
Attach Sense base clock to LPI2C15 Functional Clock.

enumerator kFRO1_DIV1_to_LPI2C15
Attach FRO1 max clock to LPI2C15 Functional Clock.

enumerator kFRO1_DIV2_to_LPI2C15
Attach FRO1 divided-by-2 to LPI2C15 Functional Clock.

enumerator kFRO2_DIV1_to_LPI2C15
Attach FRO2 max clock to LPI2C15 Functional Clock.

enumerator kNONE_to_LPI2C15
Attach NONE to LPI2C15 Functional Clock.

enumerator kSENSE_BASE_to_VDD1_CLKOUT
Attach Sense VDD1 Base Clock to VDD1 CLKOUT Clock.

enumerator kAUDIO_PLL_PFD3_to_VDD1_CLKOUT
Attach AUDIO PLL PFD3 Clock to VDD1 CLKOUT Clock.

enumerator kFRO2_DIV1_to_VDD1_CLKOUT
Attach FRO2 Max Clock to VDD1 CLKOUT Clock.

enumerator kFRO1_DIV1_to_VDD1_CLKOUT
Attach FRO1 Max Clock to VDD1 CLKOUT Clock.

enumerator kNONE_to_VDD1_CLKOUT
Attach NONE to VDD1 CLKOUT Clock.

enumerator kMEDIA_VDDN_BASE_to_MEDIA_VDDN
Attach MEDIA VDDN base clock to MEDIA VDDN clock.

enumerator kMAIN_PLL_PFD0_to_MEDIA_VDDN
Attach MAIN PLL PFD0 to MEDIA VDDN clock.

enumerator kFRO0_DIV1_to_MEDIA_VDDN
Attach FRO0 max to MEDIA VDDN clock.

enumerator kMAIN_PLL_PFD2_to_MEDIA_VDDN
Attach MAIN PLL PFD2 to MEDIA VDDN clock.

enumerator kMEDIA_VDD2_BASE_to_MEDIA_MAIN
Attach MEDIA VDD2 base clock to MEDIA MAIN clock.

enumerator kMAIN_PLL_PFD0_to_MEDIA_MAIN
Attach Main PLL PFD0 clock to MEDIA MAIN clock.

enumerator kFRO0_DIV1_to_MEDIA_MAIN
Attach FRO0 to MEDIA MAIN clock.

enumerator kMAIN_PLL_PFD2_to_MEDIA_MAIN
Attach Main PLL PFD2 clock to MEDIA MAIN clock.

enumerator kFRO1_DIV3_to_MEDIA_VDDN_BASE
Attach Fro1 divided-by-3 to Media VDDN Base Clock.

enumerator kFRO1_DIV1_to_MEDIA_VDDN_BASE
Attach Fro1 divided-by-1 to Media VDDN Base Clock.

enumerator kFRO0_DIV3_to_MEDIA_VDDN_BASE
Attach Fro0 divided-by-3 to Media VDDN Base Clock.

enumerator kLPOSC_to_MEDIA_VDDN_BASE
Attach LPOSC clock to Media VDDN Base Clock.

enumerator kFRO1_DIV3_to_MEDIA_VDD2_BASE
Attach Fro1 divided-by-3 to Media VDD2 Base Clock.

enumerator kFRO1_DIV1_to_MEDIA_VDD2_BASE
Attach Fro1 divided-by-1 to Media VDD2 Base Clock.

enumerator kFRO0_DIV3_to_MEDIA_VDD2_BASE
Attach Fro0 divided-by-3 to Media VDD2 Base Clock.

enumerator kLPOSC_to_MEDIA_VDD2_BASE
Attach LPOSC clock to Media VDD2 Base Clock.

enumerator kCOMMON_BASE_to_XSPI2
Attach Common VDDN base clock to XSPI2 Functional Clock.

enumerator kAUDIO_PLL_PFD1_to_XSPI2
Attach Audio PLL PFD1 clock to XSPI2 Functional Clock.

enumerator kFRO0_DIV1_to_XSPI2
Attach FRO0 max clock to XSPI2 Functional Clock.

enumerator kMAIN_PLL_PFD3_to_XSPI2
Attach Main PLL PFD3 clock to XSPI2 Functional Clock.

enumerator kNONE_to_XSPI2
Attach NONE to XSPI2 Functional Clock.

enumerator k32KHZ_WAKE_to_USB
Attach Wakeup 32K to USB Functional Clock.

enumerator kLPOSC_to_USB
Attach LPOSC_1M to USB Functional Clock.

enumerator kUSB_24MHZ_to_USB
Attach 24MHz USB OSC clock to USB Functional Clock.

enumerator kNONE_to_USB
Attach NONE to USB Functional Clock.

enumerator k32KHZ_WAKE_to_EUSB
Attach Wakeup 32K to eUSB Functional Clock.

enumerator kLPOSC_to_EUSB
Attach LPOSC_1M to eUSB Functional Clock.

enumerator kEUSB_24MHZ_to_EUSB
Attach 24MHz eUSB OSC clock to eUSB Functional Clock.

enumerator kNONE_to_EUSB
Attach NONE to eUSB Functional Clock.

enumerator kMEDIA_VDDN_BASE_to_SDIO0
Attach Media VDDN base clock to SDIO0 Functional Clock.

enumerator kAUDIO_PLL_PFD0_to_SDIO0
Attach Audio PLL PFD0 clock to SDIO0 Functional Clock.

enumerator kFRO0_DIV1_to_SDIO0
Attach FRO0 max clock to SDIO0 Functional Clock.

enumerator kMAIN_PLL_PFD2_to_SDIO0
Attach Main PLL PFD2 clock to SDIO0 Functional Clock.

enumerator kNONE_to_SDIO0
Attach NONE to SDIO0 Functional Clock.

enumerator kMEDIA_VDDN_BASE_to_SDIO1
Attach Media VDDN base clock to SDIO1 Functional Clock.

enumerator kAUDIO_PLL_PFD0_to_SDIO1
Attach Audio PLL PFD0 clock to SDIO1 Functional Clock.

enumerator kFRO0_DIV1_to_SDIO1
Attach FRO0 max clock to SDIO1 Functional Clock.

enumerator kMAIN_PLL_PFD1_to_SDIO1
Attach Main PLL PFD1 clock to SDIO1 Functional Clock.

enumerator kNONE_to_SDIO1
Attach NONE to SDIO1 Functional Clock.

enumerator kMEDIA_VDD2_BASE_to_MIPI_DSI_HOST_PHY
Attach Media VDD2 base clock to MIPI_DSI_Host PHY Clock.

enumerator kFRO0_DIV1_to_MIPI_DSI_HOST_PHY
Attach FRO0 max clock to MIPI_DSI_Host PHY Clock.

enumerator kAUDIO_PLL_PFD2_to_MIPI_DSI_HOST_PHY
Attach Audio PLL PFD2 clock to MIPI_DSI_Host PHY Clock.

enumerator kNONE_to_MIPI_DSI_HOST_PHY
Attach NONE to MIPI_DSI_Host PHY Clock.

enumerator kMEDIA_VDD2_BASE_to_MIPI_DPHYESC_CLK
Attach Media VDD2 base clock to MIPI_DSI_Host DPHY Escape Mode Clock.

enumerator kMAIN_PLL_PFD1_to_MIPI_DPHYESC_CLK
Attach Main PLL PFD1 clock to MIPI_DSI_Host DPHY Escape Mode Clock.

enumerator kFRO0_DIV1_to_MIPI_DPHYESC_CLK
Attach FRO0 max clock to MIPI_DSI_Host DPHY Escape Mode Clock.

enumerator kAUDIO_PLL_PFD2_to_MIPI_DPHYESC_CLK
Attach Audio PLL PFD2 clock to MIPI_DSI_Host DPHY Escape Mode Clock.

enumerator kNONE_to_MIPI_DPHYESC_CLK
Attach NONE to MIPI_DSI_Host DPHY Escape Mode Clock.

enumerator kMEDIA_VDD2_BASE_to_VGPU
Attach Media VDD2 base clock to VGPU Clock.

enumerator kMAIN_PLL_PFD0_to_VGPU
Attach Main PLL PFD0 clock to VGPU Clock.

enumerator kFRO0_DIV1_to_VGPU
Attach FRO0 max clock to VGPU Clock.

enumerator kMAIN_PLL_PFD2_to_VGPU
Attach Main PLL PFD2 clock to VGPU Clock.

enumerator kNONE_to_VGPU
Attach NONE to VGPU Clock.

enumerator kMEDIA_VDD2_BASE_to_LPSPI14
Attach Media VDD2 base clock to LPSPI14 Clock.

enumerator kFRO0_DIV1_to_LPSPI14
Attach FRO0 max clock to LPSPI14 Clock.

enumerator kMAIN_PLL_PFD0_to_LPSPI14
Attach Main PLL PFD0 clock to LPSPI14 Clock.

enumerator kFRO1_DIV1_to_LPSPI14
Attach FRO1 max clock to LPSPI14 Clock.

enumerator kNONE_to_LPSPI14
Attach NONE to LPSPI14 Clock.

enumerator kMEDIA_VDD2_BASE_to_LPSPI16
Attach Media VDD2 base clock to LPSPI16 Clock.

enumerator kFRO0_DIV1_to_LPSPI16
Attach FRO0 max clock to LPSPI16 Clock.

enumerator kMAIN_PLL_PFD0_to_LPSPI16
Attach Main PLL PFD0 clock to LPSPI16 Clock.

enumerator kFRO1_DIV1_to_LPSPI16
Attach FRO1 max clock to LPSPI16 Clock.

enumerator kNONE_to_LPSPI16
Attach NONE to LPSPI16 Clock.

enumerator kMEDIA_VDD2_BASE_to_FLEXIO
Attach MEDIA VDD2 base clock to FLEXIO.

enumerator kFRO0_DIV1_to_FLEXIO
Attach FRO0 max clock to FLEXIO.

enumerator kFRO1_DIV1_to_FLEXIO
Attach FRO1 max clock to FLEXIO.

enumerator kMAIN_PLL_PFD3_to_FLEXIO
Attach Main PLL PFD3 clock to FLEXIO.

enumerator kNONE_to_FLEXIO
Attach NONE to FLEXIO.

enumerator kMEDIA_VDD2_BASE_to_LCDIF
Attach MEDIA VDD2 base clock to LCDIF.

enumerator kMAIN_PLL_PFD2_to_LCDIF
Attach Main PLL PFD0 clock to LCDIF.

enumerator kFRO0_DIV1_to_LCDIF
Attach FRO0 max clock to LCDIF.

enumerator kAUDIO_PLL_PFD1_to_LCDIF
Attach Audio PLL PFD1 to LCDIF.

enumerator kNONE_to_LCDIF
Attach NONE to LCDIF.

enum _clock_div_name

Clock dividers.

Values:

enumerator kCLOCK_DivLPFlexComm17Clk

LP_FLEXCOMM17 Clk Divider.

enumerator kCLOCK_DivLPFlexComm18Clk

LP_FLEXCOMM18 Clk Divider.

enumerator kCLOCK_DivLPFlexComm19Clk

LP_FLEXCOMM19 Clk Divider.

enumerator kCLOCK_DivLPFlexComm20Clk

LP_FLEXCOMM20 Clk Divider.

enumerator kCLOCK_DivSenseDspClk

Sense DSP Clk Divider.

enumerator kCLOCK_DivSai3Clk

SAI3 FCLK Clk Divider.

enumerator kCLOCK_DivUtick1Clk

UTICK1 Functional Clk Divider.

enumerator kCLOCK_DivSystickClk

SYSTICK Functional Clk Divider.

enumerator kCLOCK_DivCtimer5Clk

CTIMER5 FCLK Clk Divider.

enumerator kCLOCK_DivCtimer6Clk

CTIMER6 FCLK Clk Divider.

enumerator kCLOCK_DivCtimer7Clk

CTIMER7 FCLK Clk Divider.

enumerator kCLOCK_DivI3c23Clk

I3C2 and I3C3 Functional Clk Divider.

enumerator kCLOCK_DivCommonVddnClk

Common VDDN Clk Divider.

enumerator kCLOCK_DivSenseMainClk

VDD1_SENSE Main Clock Divider.

enumerator kCLOCK_DivSenseRamClk

Senese RAM Clk Divider.

enumerator kCLOCK_DivOstimerClk
OSTIMER Clk Divider.

enumerator kCLOCK_DivSdadcClk
SDADC Clk Divider.

enumerator kCLOCK_DivAdcClk
SARADC Clk Divider.

enumerator kCLOCK_DivMicfil0Clk
DMIC0 Clk Divider.

enumerator kCLOCK_DivLpi2c15Clk
PMIC LPI2C Clk Divider.

enumerator kCLOCK_DivVdd1ClockOut
CLKOUT_VDD1 Clk Divider.

enumerator kCLOCK_DivMediaVddnClk
Media VDDN Clk Divider.

enumerator kCLOCK_DivMediaMainClk
Media Main Clk Divider.

enumerator kCLOCK_DivXspi2Clk
XSPI2 Clk Divider.

enumerator kCLOCK_DivSdio0Clk
SDIO0 Clk Divider.

enumerator kCLOCK_DivSdio1Clk
SDIO1 Clk Divider.

enumerator kCLOCK_DivDphyClk
DPHY Clk Divider.

enumerator kCLOCK_DivDphyEscRxClk
DphyEscRx Clk Divider.

enumerator kCLOCK_DivDphyEscTxClk
DphyEscTx Clk Divider.

enumerator kCLOCK_DivVgpuClk
VGPU Clk Divider.

enumerator kCLOCK_DivLpspi14Clk
LPSPI14 Clk Divider.

enumerator kCLOCK_DivLpspi16Clk
LPSPI16 Clk Divider.

enumerator kCLOCK_DivFlexioClk
FLEXIO Clk Divider.

enumerator kCLOCK_DivLcdifClk
LCDIF Pixel Clk Divider.

enumerator kCLOCK_DivLowFreqClk
Low frequency Clk Divider.

enum _clock_fro_output_en
FRO output enable.
Values:

enumerator kCLOCK_FroDiv1OutEn
Enable Fro Div1 output.

enumerator kCLOCK_FroDiv2OutEn
Enable Fro Div2 output.

enumerator kCLOCK_FroDiv3OutEn
Enable Fro Div3 output.

enumerator kCLOCK_FroDiv6OutEn
Enable Fro Div6 output.

enumerator kCLOCK_FroDiv8OutEn
Enable Fro Div8 output.

enumerator kCLOCK_FroAllOutEn
Enable all Fro output.

enum _clock_fro_interrupt
FRO Interrupt control.
Values:

enumerator kCLOCK_FroTrimUpdateReqInt
Trim Update Request Interrupt Enable.

enumerator kCLOCK_FroTuneErrInt
Tune Error Interrupt Enable.

enumerator kCLOCK_FroLossOfLockInt
Loss-of-Lock Error Interrupt Enable.

enumerator kCLOCK_FroAllIntterrupt
All Interrupt Enable.

enum _clock_fro_flag
FRO flags.
Values:

enumerator kCLOCK_FroTuneOnceDone
FRO Tune Once Done Flag.

enumerator kCLOCK_FroTrimLock
FRO Trim Lock Flag.

enumerator kCLOCK_FroTrimUpdateReq
FRO Trim Update Request Flag.

enumerator kCLOCK_FroTuneErr
FRO Tune Error Flag.

enumerator kCLOCK_FroLossOfLockErr
FRO Loss-of-lock Error Flag.

enum _clock_domain_enable
Clock Control for each power domain.
Values:

enumerator kCLOCK_Vdd2CompDomainEnable
Clock Control of VDD2_COMP Domain.

enumerator kCLOCK_Vdd1SenseDomainEnable
Clock Control of VDD1_SENSE Domain.

enumerator kCLOCK_Vdd2DspDomainEnable
Clock Control of VDD2_DSP Domain.

enumerator kCLOCK_Vdd2MediaDomainEnable
Clock Control of VDD2_MEDIA Domain.

enumerator kCLOCK_VddnMediaDomainEnable
Clock Control of VDDN_MEDIA Domain.

enumerator kCLOCK_Vdd2ComDomainEnable
Clock Control of VDD2_COM Domain.

enumerator kCLOCK_VddnComDomainEnable
Clock Control of VDDN_COM Domain.

enumerator kCLOCK_AllDomainEnable
Clock Control of all Domain.

enum _clock_pfd
PLL PFD clock name.
Values:

enumerator kCLOCK_Pfd0
PLL PFD0

enumerator kCLOCK_Pfd1
PLL PFD1

enumerator kCLOCK_Pfd2
PLL PFD2

enumerator kCLOCK_Pfd3
PLL PFD3

enum _main_pll_src
MainPLL Reference Input Clock Source.
Values:

enumerator kCLOCK_MainPllFro1Div8
FRO1_DIV8 clock

enumerator kCLOCK_MainPllOscClk
OSC clock

enum _main_pll_mult
MainPLL Multiplication Factor.
Values:

enumerator kCLOCK_MainPllMult16
Divide by 16

enumerator kCLOCK_MainPllMult17
Divide by 17

enumerator kCLOCK_MainPllMult18
Divide by 18

enumerator kCLOCK_MainPllMult19
Divide by 19

enumerator kCLOCK_MainPllMult20
Divide by 20

enumerator kCLOCK_MainPllMult21
Divide by 21

enumerator kCLOCK_MainPllMult22
Divide by 22

enum _audio_pll_src
AudioPll Reference Input Clock Source.
Values:

enumerator kCLOCK_AudioPllFro1Div8
FRO1_DIV8 clock

enumerator kCLOCK_AudioPllOscClk
OSC clock

enum _audio_pll_mult
AudioPll Multiplication Factor.
Values:

enumerator kCLOCK_AudioPllMult16
Divide by 16

enumerator kCLOCK_AudioPllMult17
Divide by 17

enumerator kCLOCK_AudioPllMult18
Divide by 18

enumerator kCLOCK_AudioPllMult19
Divide by 19

enumerator kCLOCK_AudioPllMult20
Divide by 20

enumerator kCLOCK_AudioPllMult21
Divide by 21

enumerator kCLOCK_AudioPllMult22
Divide by 22

enum _osc32k_cap_trim
Capacitor Trim Value for OSC32KNP.
Values:

enumerator kCLOCK_Osc32kCapPf0
Capacitor Trim Value 0pF.

enumerator kCLOCK_Osc32kCapPf2
Capacitor Trim Value 2pF.

enumerator kCLOCK_Osc32kCapPf4
Capacitor Trim Value 4pF.

enumerator kCLOCK_Osc32kCapPf6
Capacitor Trim Value 6pF.

enumerator kCLOCK_Osc32kCapPf8
Capacitor Trim Value 8pF.

enumerator kCLOCK_Osc32kCapPf10
Capacitor Trim Value 10pF.

enumerator kCLOCK_Osc32kCapPf12
Capacitor Trim Value 12pF.

enumerator kCLOCK_Osc32kCapPf14
Capacitor Trim Value 14pF.

enumerator kCLOCK_Osc32kCapPf16
Capacitor Trim Value 16pF.

enumerator kCLOCK_Osc32kCapPf18
Capacitor Trim Value 18pF.

enumerator kCLOCK_Osc32kCapPf20
Capacitor Trim Value 20pF.

enumerator kCLOCK_Osc32kCapPf22
Capacitor Trim Value 22pF.

enumerator kCLOCK_Osc32kCapPf24
Capacitor Trim Value 24pF.

enumerator kCLOCK_Osc32kCapPf26
Capacitor Trim Value 26pF.

enumerator kCLOCK_Osc32kCapPf28
Capacitor Trim Value 28pF.

enumerator kCLOCK_Osc32kCapPf30
Capacitor Trim Value 30pF.

typedef enum *_clock_usb_src* clock_usb_src_t
USB clock source definition.

typedef enum *_clock_usb_phy_src* clock_usb_phy_src_t

Source of the USB HS PHY.

typedef enum *_clock_ip_name* clock_ip_name_t

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

typedef enum *_clock_name* clock_name_t

Clock name used to get clock frequency.

typedef enum *_clock_attach_id* clock_attach_id_t

The enumerator of clock attach Id for multi sources clock slices. Not all Muxer has MUX output enable control, if no mux output enable control, the mux output will be always enabled.

typedef enum *_clock_div_name* clock_div_name_t

Clock dividers.

typedef enum *_clock_fro_output_en* clock_fro_output_en_t

FRO output enable.

typedef struct *_clock_fro_config* clock_fro_config_t

FRO configuration.

typedef enum *_clock_domain_enable* clock_domain_enable_t

Clock Control for each power domain.

typedef enum *_clock_pfd* clock_pfd_t

PLL PFD clock name.

typedef enum *_main_pll_src* main_pll_src_t

MainPLL Reference Input Clock Source.

typedef enum *_main_pll_mult* main_pll_mult_t

MainPLL Multiplication Factor.

typedef struct *_clock_main_pll_config* clock_main_pll_config_t

PLL configuration for MAINPLL.

typedef enum *_audio_pll_src* audio_pll_src_t

AudioPll Reference Input Clock Source.

typedef enum *_audio_pll_mult* audio_pll_mult_t

AudioPll Multiplication Factor.

typedef struct *_clock_audio_pll_config* clock_audio_pll_config_t

PLL configuration for Audio PLL.

typedef enum *_osc32k_cap_trim* osc32k_cap_trim_t

Capacitor Trim Value for OSC32KNP.

typedef struct *_clock_osc32k_config_t* clock_osc32k_config_t

configuration for 32K OSC

volatile uint32_t g_xtalFreq

External XTAL (SYSOSC) clock frequency.

The XTAL (SYSOSC) clock frequency in Hz, when the clock is setup, use the function CLOCK_SetXtalFreq to set the value in to clock driver. For example, if XTAL is 16MHz,

```
CLOCK_SetXtalFreq(160000000);
```

volatile uint32_t g_clkinFreq

External CLK_IN pin clock frequency (clkin).

The CLK_IN pin (clkin) clock frequency in Hz, when the clock is setup, use the function CLOCK_SetClkinFreq to set the value in to clock driver. For example, if CLK_IN is 16MHz,

```
CLOCK_SetClkinFreq(160000000);
```

volatile uint32_t g_32kClkinFreq

External 32KHz input clock frequency.

The External 32KHz input clock frequency in Hz, used when OSC32KNP in bypass mode, use the function CLOCK_Set32kClkinFreq to set the value in to clock driver. For example, if EXTALIN32K input is 32768,

```
CLOCK_Set32kClkinFreq(32768);
```

volatile uint32_t g_mclkFreq

External MCLK IN clock frequency.

The MCLK IN clock frequency in Hz, when the clock is setup, use the function CLOCK_SetMclkFreq to set the value in to clock driver. For example, if MCLK IN is 16MHz,

```
CLOCK_SetMclkFreq(160000000);
```

void CLOCK_EnableClock(*clock_ip_name_t* clk)

Enable the clock for specific IP.

Parameters

- clk – : Clock to be enabled.

Returns

Nothing

void CLOCK_DisableClock(*clock_ip_name_t* clk)

Disable the clock for specific IP.

Parameters

- clk – : Clock to be disabled.

Returns

Nothing

void CLOCK_AttachClk(*clock_attach_id_t* connection)

brief Configure the clock selection muxes.

For some of the muxes, there's SEL_EN bit to gate the mux ouput to reduce power, using kNONT_to_XXX to gate the mux oupput. used by any peripheral.

Parameters

- connection – : Clock to be configured.

Returns
Nothing

void CLOCK_SetClkDiv(*clock_div_name_t* div_name, uint32_t divider)

Setup peripheral clock dividers.

Parameters

- div_name – : Clock divider name
- divider – : Value to be divided. Divided clock frequency = Undivided clock frequency / divider.

Returns
Nothing

uint32_t CLOCK_GetFreq(*clock_name_t* clockName)

Return Frequency of selected clock.

Returns
Frequency of selected clock

uint32_t CLOCK_GetSenseAudioClkFreq(void)

Return Frequency of VDD1 audio clk.

Returns
Frequency of audio_clk

uint32_t CLOCK_GetHifi1ClkFreq(void)

Return Frequency of Sense DSP clk.

Returns
Frequency of Sense DSP clk

void CLOCK_EnableFroClkOutput(FRO_Type *base, uint32_t divOutEnable)

Enable/Disable FRO clock output.

Parameters

- base – : base address of FRO.
- divOutEnable – : Or'ed value of *clock_fro_output_en_t* to enable certain clock freq output.

void CLOCK_ConfigFroTrim(FRO_Type *base, uint16_t trimVal)

Configure FRO trim values when FRO is configured in Open loop mode.

Parameters

- base – : base address of FRO.
- trimVal – : 12bits trim value.

void CLOCK_DisableFro(FRO_Type *base)

Disable the FRO clock. This API will disable the FRO clock output and power off FRO.

Parameters

- base – : base address of FRO.

status_t CLOCK_EnableFroAutoTuning(FRO_Type *base, const *clock_fro_config_t* *config, bool enable)

Enable/Disable FRO for close loop mode(autotuner).

Parameters

- base – : base address of FRO.
- config – : The configuration for FRO.
- enable – Enable auto tuning mode or not.

Return values

- kStatus_Success – successfully tuned to the target configuration.
- kStatus_InvalidArgument – Invalid arguement.
- kStatus_Fail – failed to lock to the target frequency.

void CLOCK_EnableFroClkFreq(FRO_Type *base, uint32_t targetFreq, uint32_t divOutEnable)

Enable FRO clock output with specified frequency.

Parameters

- base – : base address of FRO.
- targetFreq – target fro frequency.
- divOutEnable – Or'ed value of *clock_fro_output_en_t* to enable certain clock freq output.

status_t CLOCK_EnableFroClkFreqCloseLoop(FRO_Type *base, const *clock_fro_config_t* *config, uint32_t divOutEnable)

Enable FRO clock output with target frequency using FRO close loop mode. For example, to enable FRO2 to ouput 200MHZ:

```
  const clock_fro_config_t config = {
   .targetFreq = 200000000U,
   .range = 100U,
   .trim1DelayUs = 15U,
   .trim2DelayUs = 150U,
   .refDiv = 0U,
   .enableInt = 0U,
   .coarseTrimEn = true,
};
CLOCK_EnableFroClkFreqCloseLoop(FRO2, &config, kCLOCK_FroAllOutEn);
```

If the API returns kStatus_Timeout or kStatus_Fail, the CLOCK_FroTuneOnce and trim value from AUTOTRIM register can be used to get the desired frequency in open loop mode.

Parameters

- base – : base address of FRO.
- config – : The configuration for FRO.
- divOutEnable – : Or'ed value of clock_fro_output_en_t to enable certain clock freq output.

Return values

- kStatus_Success – successfully tuned to the target configuration.
- kStatus_InvalidArgument – Invalid arguement.
- kStatus_Timeout – Timeout to lock to the target frequency. The config->range may too strict(Suggested range >=100) or the auto tuner can't lock to the desired frequency.
- kStatus_Fail – The FRO locked with error.

uint32_t CLOCK_FroTuneOnce(FRO_Type *base, uint16_t trimVal)

FRO Tune once. Use FRO tuneonce feature to calculate the FRO output frequency with given trimVal. The FRO should be powered up and reference clock(SOSC) should be enabled.

Parameters

- base – : base address of FRO.
- trimVal – : trim value.

Returns

FRO frequency in Hz.

void CLOCK_FroFineTune(FRO_Type *base, uint32_t targetFreq, uint16_t trimVal)

Fine tune FRO output frequency in open loop mode. The FRO should be powered up and reference clock(SOSC) should be enabled. The API uses the FRO tuneonce feature to try trim values from (trimVal-1, trimVal, trimVal+1) and select a best one.

Parameters

- base – : base address of FRO.
- targetFreq – : Target frequency in Hz.
- trimVal – : trim value. When CLOCK_EnableFroClkFreqCloseLoop failed to lock, the value in FRO AUTOTRIM register can be used to do fine tune.

uint32_t CLOCK_GetFroFlags(FRO_Type *base)

Get FRO flags.

Parameters

- base – : base address of FRO.
- flags – Or'ed value of *_clock_fro_flag*.

static inline void CLOCK_ClearFroFlags(FRO_Type *base, uint32_t flags)

Clear FRO flags.

Parameters

- base – : base address of FRO.
- flags – Or'ed value of *_clock_fro_flag* to clear.

static inline void CLOCK_EnableFroFastStartup(FRO_Type *base, bool enable)

Enable/Disable FRO fast startup.

Parameters

- base – : base address of FRO.
- true (enable,) – to enable fast startup, false to disable fast startup.

uint32_t CLOCK_GetFroClkFreq(uint32_t id)

Return Frequency of FRO clk.

Parameters

- id – FRO index

Returns

Frequency of FRO clk

uint32_t CLOCK_GetSenseMainClkFreq(void)

Return Frequency of sense main clk.

Returns

Frequency of main clk

uint32_t CLOCK_GetMainPllFreq(void)

Return Frequency of MAINPLL.

Returns
Frequency of MAINPLL

uint32_t CLOCK_GetMainPfdFreq(*clock_pfd_t* pfd)

Get current output frequency of specific Main PLL PFD.

Parameters

- pfd – : pfd name to get frequency.

Returns
Frequency of MainPLL PFD.

uint32_t CLOCK_GetAudioPllFreq(void)

Return Frequency of AUDIO PLL.

Returns
Frequency of AUDIO PLL

uint32_t CLOCK_GetAudioPfdFreq(*clock_pfd_t* pfd)

Get current output frequency of specific Audio PLL PFD.

Parameters

- pfd – : pfd name to get frequency.

Returns
Frequency of AUDIO PLL PFD.

static inline uint32_t CLOCK_GetXtalInClkFreq(void)

Return Frequency of High-Freq output of FRO.

Return Frequency of sys osc Clock

Returns
Frequency of High-Freq output of FRO

Returns
Frequency of sys osc Clock. Or CLK_IN pin frequency.

uint32_t CLOCK_GetSenseBaseClkFreq(void)

Return Frequency of Sense Base clock.

Returns
Frequency of sense base clk

void CLOCK_EnableSysOscClk(bool enable, bool enableLowPower, uint32_t delay_us)

Enable/Disable sys osc clock from external crystal clock.

Parameters

- enable – : true to enable system osc clock, false to bypass system osc.
- enableLowPower – : true to enable low power mode, false to enable high gain mode.
- delay_us – : Delay time after OSC power up.

uint32_t CLOCK_GetVddnComBaseClkFreq(void)

Return Frequency of VDDN_COM Base Clock.

Returns
Frequency of baseclk_comn

uint32_t CLOCK_GetVddnMediaBaseClkFreq(void)

Return Frequency of VDDN_MEDIA Base Clock.

Returns

Frequency of baseclk_mdn

uint32_t CLOCK_GetVdd2MediaBaseClkFreq(void)

Return Frequency of VDD2_MEDIA Base Clock.

Returns

Frequency of baseclk_md2

uint32_t CLOCK_GetMediaMainClkFreq(void)

Return Frequency of MEDIA_MAIN Clock.

Returns

Frequency of media_main_clk

uint32_t CLOCK_GetMediaVddnClkFreq(void)

Return Frequency of MEDIA_VDDN Clock.

Returns

Frequency of media_vddn_clk

void CLOCK_EnableFro2ClkForDomain(uint32_t domainEnable)

Enable/Disable FRO2 clock for various domains.

Parameters

- domainEnable – : Or'ed value of *clock_domain_enable_t* to enable DRFO2 max clock for certain domain.

void CLOCK_EnableMainPllPfdClkForDomain(*clock_pfd_t* pfd, uint32_t domainEnable)

Enable/Disable MainPLL PFD clock for various domains.

Enables PFD clock of MainPLL for various domains. Each PFD clock of MainPLL can be sent to various domain. To reduce power consumption, turn off the PFD in the domain when this PFD is not configured.

Parameters

- pfd – : Which PFD clock to control.
- domainEnable – : Or'ed value of *clock_domain_enable_t* to enable clock for certain domain.

void CLOCK_EnableAudioPllPfdClkForDomain(*clock_pfd_t* pfd, uint32_t domainEnable)

Enable/Disable AUDIO PFD clock for various domains.

Enables PFD clock of AUDIO for various domains. Each PFD clock of AUDIO can be sent to various domain. To reduce power consumption, turn off the PFD in the domain when this PFD is not configured.

Parameters

- pfd – : Which PFD clock to control.
- domainEnable – : Or'ed value of *clock_domain_enable_t* to enable clock for certain domain.

void CLOCK_EnableAudioPllVcoClkForDomain(uint32_t domainEnable)

Enable/Disable AUDIO VCO clock for various domains.

VCO clock of Audio PLL can be sent to various domains to reduce power consumption when VCO is not set to use in such domain. This VCO can be disabled for such domain.

Parameters

- domainEnable – : Or'ed value of *clock_domain_enable_t* to enable clock for certain domain.

void CLOCK_InitMainPll(const *clock_main_pll_config_t* *config)

Initialize the Main PLL.

Parameters

- config – : Configuration to set to PLL.

static inline void CLOCK_DeinitMainPll(void)

Deinit the Main PLL.

Parameters

- none. –

status_t CLOCK_InitMainPfd(*clock_pfd_t* pfd, uint8_t divider)

Initialize the Main PLL PFD.

Note: It is recommended that PFD settings are kept between 12-35.

Parameters

- pfd – : Which PFD clock to enable.
- divider – : The PFD divider value.

Returns

kStatus_Success if successfully, kStatus_Timeout if timeout happen.

static inline void CLOCK_DeinitMainPfd(*clock_pfd_t* pfd)

Disable the Main PLL PFD.

Parameters

- pfd – : Which PFD clock to disable.

void CLOCK_InitAudioPll(const *clock_audio_pll_config_t* *config)

Initialize the audio PLL.

Parameters

- config – : Configuration to set to PLL.

static inline void CLOCK_DeinitAudioPll(void)

Deinit the Audio PLL.

Parameters

- none. –

status_t CLOCK_InitAudioPfd(*clock_pfd_t* pfd, uint8_t divider)

Initialize the audio PLL PFD.

Note: It is recommended that PFD settings are kept between 12-35.

Parameters

- pfd – : Which PFD clock to enable.
- divider – : The PFD divider value.

Returns

kStatus_Success if successfully, kStatus_Timeout if timeout happen.

static inline void CLOCK_DeinitAudioPfd(*clock_pfd_t* pfd)

Disable the audio PLL PFD.

Parameters

- pfd – : Which PFD clock to disable.

static inline uint32_t CLOCK_GetLpOscFreq(void)

Return Frequency of Lower power osc.

Returns

Frequency of LPOSC

static inline uint32_t CLOCK_GetSysOscFreq(void)

Return Frequency of sys osc Clock.

Returns

Frequency of sys osc Clock. Or CLK_IN pin frequency.

static inline uint32_t CLOCK_GetMclkInClkFreq(void)

Return Frequency of MCLK Input Clock.

Returns

Frequency of MCLK input Clock.

uint32_t CLOCK_GetOsc32KFreq(void)

Return Frequency of 32kHz osc.

Returns

Frequency of 32kHz osc

void CLOCK_EnableOsc32K(*clock_osc32k_config_t* *config)

Enables OSC32KNP.

Parameters

- config – : configuration set to OSC32KNP

static inline void CLOCK_DisableOsc32K(void)

Disable OSC32KNP.

uint32_t CLOCK_GetWakeClk32KFreq(void)

Return Frequency of 32khz wake clk.

Returns

Frequency of 32kHz wake clk

static inline void CLOCK_SetXtalFreq(uint32_t freq)

Set the XTALIN (system OSC) frequency based on board setting. NOTE, when SOSC is used, either CLOCK_SetXtalFreq or CLOCK_SetClkinFreq(But NOT both.) should be called to tell driver the clock frequency connected to SOSC module.

Parameters

- freq – : The XTAL input clock frequency in Hz.

static inline void CLOCK_SetClkinFreq(uint32_t freq)

Set the CLKIN (CLKIN pin) frequency based on board setting.

Parameters

- freq – : The CLK_IN pin input clock frequency in Hz.

static inline void CLOCK_Set32kClkinFreq(uint32_t freq)

Set the 32KHz external input frequency based on board setting.

Parameters

- freq – : The 32KHz external pin input clock frequency in Hz.

static inline void CLOCK_SetMclkFreq(uint32_t freq)

Set the MCLK IN frequency based on board setting.

Parameters

- freq – : The MCLK input clock frequency in Hz.

static inline uint32_t CLOCK_GetCoreSysClkFreq(void)

Return Frequency of Core/system clock.

Returns

Frequency of core or system Clock

uint32_t CLOCK_GetXspiClkFreq(uint32_t id)

Return Frequency of XSPI function clock.

Parameters

- id – : XSPI index to get frequency.

Returns

Frequency of XSPI functional Clock

uint32_t CLOCK_GetUtickClkFreq(void)

Return Frequency of UTICK function clock.

Returns

Frequency of UTICK functional Clock

uint32_t CLOCK_GetSystickClkFreq(void)

Return Frequency of systick clk.

Returns

Frequency of systick clk

uint32_t CLOCK_GetWdtClkFreq(uint32_t id)

Return Frequency of WDT clk.

Parameters

- id – : WDT index to get frequency.

Returns

Frequency of WDT clk

uint32_t CLOCK_GetAcmpClkFreq(void)

Return Frequency of ACMP clk.

Returns

Frequency of ACMP clk

uint32_t CLOCK_GetSaiClkFreq(void)

Return Frequency of SAI clk.

Returns

Frequency of SAI clk

uint32_t CLOCK_GetUsbClkFreq(void)

Return Frequency of USB clk.

Returns

Frequency of USB clk

uint32_t CLOCK_GetI3cClkFreq(void)

Return Frequency of I3C clk.

Returns

Frequency of I3C clk

uint32_t CLOCK_GetLPFlexCommClkFreq(uint32_t id)

Return Frequency of Flexcomm functional Clock.

Parameters

- id – : flexcomm index to get frequency.

Returns

Frequency of Flexcomm functional Clock

uint32_t CLOCK_GetLPI2cClkFreq(uint32_t id)

Return Frequency of LPI2C functional Clock.

Parameters

- id – : LPI2C index to get frequency.

Returns

Frequency of LPI2C functional Clock

uint32_t CLOCK_GetLPSpiClkFreq(uint32_t id)

Return Frequency of LPSPI functional Clock.

Parameters

- id – : LPSPI index to get frequency.

Returns

Frequency of LPSPI functional Clock

uint32_t CLOCK_GetFlexioClkFreq(void)

Return Frequency of Flexio functional Clock.

Returns

Frequency of Flexcomm functional Clock

uint32_t CLOCK_GetOSTimerClkFreq(void)

Return Frequency of OSTIMER functional Clock.

Returns

Frequency of OSTIMER functional Clock

uint32_t CLOCK_GetMicfilClkFreq(void)

Return Frequency of MICFIL functional Clock.

Returns

Frequency of MICFIL functional Clock

uint32_t CLOCK_GetCTimerClkFreq(uint32_t id)

Return Frequency of Ctimer Clock.

Parameters

- id – : ctimer index to get frequency.

Returns

Frequency of Ctimer Clock

uint32_t CLOCK_GetVdd1ClockOutClkFreq(void)

Return Frequency of VDD1 Clock Out.

Returns

Frequency of ClockOut of sense domain

uint32_t CLOCK_GetAdcClkFreq(void)

Return Frequency of Adc Clock.

Returns

Frequency of Adc Clock.

uint32_t CLOCK_GetSdAdcClkFreq(void)

Return Frequency of SDADC Clock.

Returns

Frequency of SDADC Clock.

uint32_t CLOCK_GetVgpuClkFreq(void)

Return Frequency of VGPU functional Clock.

Returns

Frequency of VGPU functional Clock

uint32_t CLOCK_GetLcdifClkFreq(void)

Return Frequency of LCDIF pixel Clock.

Returns

Frequency of LCDIF pixel Clock

uint32_t CLOCK_GetMipiDphyClkFreq(void)

Return Frequency of MIPI DPHY functional Clock.

Returns

Frequency of MIPI DPHY functional Clock

uint32_t CLOCK_GetMipiDphyEscRxClkFreq(void)

Return Frequency of MIPI DPHY Esc RX functional Clock.

Returns

Frequency of MIPI DPHY Esc RX functional Clock

uint32_t CLOCK_GetMipiDphyEscTxClkFreq(void)

Return Frequency of MIPI DPHY Esc Tx functional Clock.

Returns

Frequency of MIPI DPHY Esc Tx functional Clock

uint32_t CLOCK_GetUsdhcClkFreq(uint32_t id)

Return Frequency of USDHC Clock.

Parameters

- id – : uSDHC index to get frequency.

Returns

Frequency of USDHC Clock

bool CLOCK_EnableUsbhs0PhyPllClock(*clock_usb_phy_src_t* src, uint32_t freq)

Enable USB HS PHY PLL clock.

This function enables the internal 480MHz USB PHY PLL clock.

Parameters

- src – USB HS PHY PLL clock source.
- freq – The frequency specified by src.

Return values

- true – The clock is set successfully.
- false – The clock source is invalid to get proper USB HS clock.

bool CLOCK_EnableUsbhs0Clock(*clock_usb_src_t* src, uint32_t freq)

Enable USB HS clock.

This function only enables the access to USB HS prepheral, upper layer should first call the ref CLOCK_EnableUsbhs0PhyPllClock to enable the PHY clock to use USB HS.

Parameters

- src – USB HS does not care about the clock source, here must be ref kCLOCK_UsbSrcUnused.
- freq – USB HS does not care about the clock source, so this parameter is ignored.

Return values

- true – The clock is set successfully.
- false – The clock source is invalid to get proper USB HS clock.

void CLOCK_DisableUsbhs0PhyPllClock(void)

Disable USB HS PHY PLL clock.

This function disables USB HS PHY PLL clock.

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.5.1.

CLOCK_DeinitSysPfd

CLOCK_InitSysPfd

CDOG_CLOCKS

Clock ip name array for CDOG.

FREQME_CLOCKS

Clock ip name array for FREQME.

GPU_CLOCKS

Clock ip name array for GPU.

MIPI_DSI_HOST_CLOCKS

Clock ip name array for MIPI DSI.

LCDIF_CLOCKS

Clock ip name array for LCDIF.

SCT_CLOCKS

Clock ip name array for SCT.

USB_CLOCKS

Clock ip name array for USB.

USBPHY_CLOCKS

Clock ip name array for USBPHY.

XSPI_CLOCKS
: Clock ip name array for FlexSPI.

MMU_CLOCKS
: Clock ip name array for MMU.

CACHE64_CLOCKS
: Clock ip name array for CACHE64.

CACHE64POLSEL_CLOCKS
: Clock ip name array for CACHE64_POLSEL.

LPADC_CLOCKS
: Clock ip name array for ADC.

SDADC_CLOCKS
: Clock ip name array for SDADC.

CMP_CLOCKS
: Clock ip name array for ACMP.

USDHC_CLOCKS
: Clock ip name array for uSDHC.

WWDT_CLOCKS
: Clock ip name array for WWDT.

UTICK_CLOCKS
: Clock ip name array for UTICK.

FLEXIO_CLOCKS
: Clock ip name array for FlexIO.

LP_FLEXCOMM_CLOCKS
: Clock ip name array for LP_FLEXCOMM.

LPUART_CLOCKS
: Clock ip name array for LPUART.

LPI2C_CLOCKS
: Clock ip name array for LPI2C.

LPSPI_CLOCKS
: Clock ip name array for LSPI.

SAI_CLOCKS
: Clock ip name array for SAI.

SEMA42_CLOCKS
: Clock ip name array for SEMA.

CRC_CLOCKS
: Clock ip name array for MU.

  Clock ip name array for CRC.

GPIO_CLOCKS
: Clock ip name array for GPIO.

PDM_CLOCKS
: Clock ip name array for PDM.

PINT_CLOCKS
: Clock ip name array for PINT.

PNGDEC_CLOCKS
: Clock ip name array for PNGDEC.

JPEGDEC_CLOCKS
: Clock ip name array for JPEGDEC.

I3C_CLOCKS
: Clock ip name array for I3C.

MRT_CLOCKS
: Clock ip name array for MRT.

CTIMER_CLOCKS
: Clock ip name array for CT32B.

OSTIMER_CLOCKS
: Clock ip name array for OSTIMER.

RTC_CLOCKS
: Clock ip name array for RTC.

CLK_GATE_REG_OFFSET_SHIFT
: Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

CLK_GATE_REG_OFFSET_MASK

CLK_GATE_BIT_SHIFT_SHIFT

CLK_GATE_BIT_SHIFT_MASK

CLK_GATE_DEFINE(reg_offset, bit_shift)

CLK_GATE_ABSTRACT_REG_OFFSET(x)

CLK_GATE_ABSTRACT_BITS_SHIFT(x)

CLK_CTL0_PSCCTL0

CLK_CTL0_PSCCTL1

CLK_CTL0_PSCCTL2

CLK_CTL0_PSCCTL3

CLK_CTL0_PSCCTL4

CLK_CTL0_PSCCTL5

CLK_CTL1_PSCCTL0

CLK_CTL1_PSCCTL1

CLK_CTL2_PSCCTL0

CLK_CTL3_PSCCTL0

CLK_CTL4_PSCCTL0

CLK_CTL4_PSCCTL1

SYSCON3_SEC_CLK_CTRL

CLKCTL0_ONE_SRC_CLKSLICE_ENABLE

CLKCTL3_ONE_SRC_CLKSLICE_ENABLE

CLKCTL4_ONE_SRC_CLKSLICE_ENABLE

CLK_MUX_INST_INDEX_SHIFT

Clock Mux Switches The encoding is as follows each connection identified is 32bits wide starting from LSB upwards.

[31-29 14 12:11 10:0] [CLKCTL index]: [Disable MUX output]: [MUXA choice]:[MUXA offset]. MUX offset 0 means end of descriptor.

CLK_MUX_CHOICE_OFFSET_SHIFT

CLK_MUX_DISABLE_OUTPUT_MASK

SENSEBASECLKSEL_OFFSET

SENSEDSPCPUCLKDIV_OFFSET

SENSEDSPCPUCLKSEL_OFFSET

SAI3FCLKSEL_OFFSET

SAI3CLKDIV_OFFSET

UTICK1FCLKSEL_OFFSET

UTICK1CLKDIV_OFFSET

WWDT2FCLKSEL_OFFSET

WWDT3FCLKSEL_OFFSET

SYSTICKFCLKSEL_OFFSET

SYSTICKFCLKDIV_OFFSET

CTIMER5FCLKSEL_OFFSET

CTIMER6FCLKSEL_OFFSET

CTIMER7FCLKSEL_OFFSET

CTIMER5CLKDIV_OFFSET

CTIMER6CLKDIV_OFFSET

CTIMER7CLKDIV_OFFSET

I3C23FCLKSEL_OFFSET

I3C23FCLKDIV_OFFSET

FC17FCLKSEL_OFFSET

FC18FCLKSEL_OFFSET

FC19FCLKSEL_OFFSET

FC20FCLKSEL_OFFSET

FC17FCLKDIV_OFFSET

FC18FCLKDIV_OFFSET

FC19FCLKDIV_OFFSET

FC20FCLKDIV_OFFSET

AUDIOVDD1CLKSEL_OFFSET

USBCLKSRC24MCLKSEL_OFFSET

USBCLKSRC24MCLKSEL_OFFSET

COMMONVDDNCLKSEL_OFFSET

COMMONVDDNCLKDIV_OFFSET

COMNBASECLKSEL_OFFSET

EUSBCLKSRC24MCLKSEL_OFFSET

MAINPLL0CLKSEL_OFFSET

MAINPLL0LOCKTIMEDIV2_OFFSET

AUDIOPLL0CLKSEL_OFFSET

AUDIOPLL0LOCKTIMEDIV2_OFFSET

SENSEMAINCLKDIV_OFFSET

SENSE_MAINCLKSEL_OFFSET

SENSERAMCLKSEL_OFFSET

SENSERAMCLKDIV_OFFSET

OSEVENTTFCLKSEL_OFFSET

OSEVENTFCLKDIV_OFFSET

SDADCFCLKSEL_OFFSET

SDADCFCLKDIV_OFFSET

SARADCFCLKSEL_OFFSET

SARADCFCLKDIV_OFFSET

WAKE32KCLKSEL_OFFSET

MICFIL0FCLKSEL_OFFSET

MICFIL0FCLKDIV_OFFSET

LPI2CFCLKSEL_OFFSET

LPI2CFCLKDIV_OFFSET

SENSE_CLKOUTCLKSEL_OFFSET

SENSE_CLKOUTCLKDIV_OFFSET

MEDIAVDDNCLKSEL_OFFSET

MEDIAVDDNCLKDIV_OFFSET

MEDIAMAINCLKSEL_OFFSET

MEDIAMAINCLKDIV_OFFSET

MDNBASECLKSEL_OFFSET

MD2BASECLKSEL_OFFSET

XSPI2FCLKSEL_OFFSET

XSPI2FCLKDIV_OFFSET

USBFCLKSEL_OFFSET

EUSBFCLKSEL_OFFSET

SDIO0FCLKSEL_OFFSET

SDIO0FCLKDIV_OFFSET

SDIO1FCLKSEL_OFFSET

SDIO1FCLKDIV_OFFSET

DPHYCLKSEL_OFFSET

DPHYCLKDIV_OFFSET

DPHYESCCLKSEL_OFFSET

DPHYESCRXCLKDIV_OFFSET

DPHYESCTXCLKDIV_OFFSET

VGPUCLKSEL_OFFSET

VGPUCLKDIV_OFFSET

LPSPI14CLKSEL_OFFSET

LPSPI14CLKDIV_OFFSET

LPSPI16CLKSEL_OFFSET

LPSPI16CLKDIV_OFFSET

FLEXIOCLKSEL_OFFSET

FLEXIOCLKDIV_OFFSET

LCDIFPIXELCLKSEL_OFFSET

LCDIFPIXELCLKDIV_OFFSET

LOWFREQCLKDIV_OFFSET

CLKCTL0_INDEX

CLKCTL1_INDEX

CLKCTL2_INDEX

CLKCTL3_INDEX

CLKCTL4_INDEX

CLKCTL0_TUPLE_MUXA(reg, choice)

CLKCTL1_TUPLE_MUXA(reg, choice)

CLKCTL2_TUPLE_MUXA(reg, choice)

CLKCTL3_TUPLE_MUXA(reg, choice)

CLKCTL4_TUPLE_MUXA(reg, choice)

CLKCTL0_TUPLE_MUXA_NONE(reg, choice)

Macro for gated clock mux

CLKCTL1_TUPLE_MUXA_NONE(reg, choice)

CLKCTL2_TUPLE_MUXA_NONE(reg, choice)

CLKCTL3_TUPLE_MUXA_NONE(reg, choice)

CLKCTL4_TUPLE_MUXA_NONE(reg, choice)

CLKCTL_TUPLE_REG(base, tuple)

CLKCTL_TUPLE_SEL(tuple)

uint32_t targetFreq

Target frequency.

uint16_t refDiv

OSC Reference clock divider. 1 for divide ratio 1.

uint16_t trim1DelayUs

Trim 1 delay in us, minimum is 15us. Used when running in Closed Loop mode and trim values are updated by 1 decimal unit.

uint16_t trim2DelayUs

Trim 2 delay in us. Used at start of closed loop mode when auto tuner is updating trim values by 16 decimal unit steps

uint8_t range

Trim Expected Count Range. Specifies the + or - counts that the FRO frequency can be off from TEXPCNT to be considerred locked. The value/100 is the % deviation.

uint32_t enableInt

Enable interrupts. Bit mask of *_clock_fro_interrupt*.

bool coarseTrimEn

Coarse Trim Enable. Set to true to allow autotrimming of the FRO high-byte trim bits.

bool fastStartupEn

Fast startup Enable. Enables FRO fast startup.

main_pll_src_t main_pll_src

Reference Input Clock Source.

uint32_t numerator

30 bit numerator of fractional loop divider.

uint32_t denominator

30 bit numerator of fractional loop divider.

main_pll_mult_t main_pll_mult
: Multiplication Factor.

audio_pll_src_t audio_pll_src
: Reference Input Clock Source.

uint32_t numerator
: 30 bit numerator of fractional loop divider.

uint32_t denominator
: 30 bit numerator of fractional loop divider.

audio_pll_mult_t audio_pll_mult
: Multiplication Factor.

bool enableVcoOut
: Enable VCO output

bool bypass
: Bypass enable.

bool monitorEnable
: Clock Monitor Enable.

bool lowPowerMode
: Low-Power (Nano-Power) mode enable. NOTE, can only change from High-Power mode to Nano-Power mode, and not vice versa.

osc32k_cap_trim_t cap
: Capacitor Trim Value.

struct _clock_fro_config
: *#include <fsl_clock.h>* FRO configuration.

struct _clock_main_pll_config
: *#include <fsl_clock.h>* PLL configuration for MAINPLL.

struct _clock_audio_pll_config
: *#include <fsl_clock.h>* PLL configuration for Audio PLL.

struct _clock_osc32k_config_t
: *#include <fsl_clock.h>* configuration for 32K OSC

2.5 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION

CRC driver version. Version 2.1.0.

Current version: 2.1.0

Change log:

- Version 2.1.0
  - Choosing CRC clocks from CRC clock array according to instance instead of hard-coded value.
- Version 2.0.5
  - Fix CERT-C issue with boolean-to-unsigned integer conversion.
- Version 2.0.4
  - Release peripheral from reset if necessary in init function.
- Version 2.0.3
  - Fix MISRA issues
- Version 2.0.2
  - Fix MISRA issues
- Version 2.0.1
  - move DATA and DATALL macro definition from header file to source file

enum _crc_bits

CRC bit width.

Values:

enumerator kCrcBits16

Generate 16-bit CRC code

enumerator kCrcBits32

Generate 32-bit CRC code

enum _crc_result

CRC result type.

Values:

enumerator kCrcFinalChecksum

CRC data register read value is the final checksum. Reflect out and final xor protocol features are applied.

enumerator kCrcIntermediateChecksum

CRC data register read value is intermediate checksum (raw value). Reflect out and final xor protocol feature are not applied. Intermediate checksum can be used as a seed for *CRC_Init()* to continue adding data to this checksum.

typedef enum *_crc_bits* crc_bits_t

CRC bit width.

typedef enum *_crc_result* crc_result_t

CRC result type.

typedef struct *_crc_config* crc_config_t

CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

void CRC_Init(CRC_Type *base, const *crc_config_t* *config)

Enables and configures the CRC peripheral module.

This function enables the clock gate in the SIM module for the CRC peripheral. It also configures the CRC module and starts a checksum computation by writing the seed.

Parameters

- base – CRC peripheral address.
- config – CRC module configuration structure.

void CRC_Deinit(CRC_Type *base)

Disables the CRC peripheral module.

This function disables the clock gate in the SIM module for the CRC peripheral.

Parameters

- base – CRC peripheral address.

void CRC_GetDefaultConfig(*crc_config_t* *config)

Loads default values to the CRC protocol configuration structure.

Loads default values to the CRC protocol configuration structure. The default values are as follows.

```
config->polynomial = 0x1021;
config->seed = 0xFFFF;
config->reflectIn = false;
config->reflectOut = false;
config->complementChecksum = false;
config->crcBits = kCrcBits16;
config->crcResult = kCrcFinalChecksum;
```

Parameters

- config – CRC protocol configuration structure.

void CRC_WriteData(CRC_Type *base, const uint8_t *data, size_t dataSize)

Writes data to the CRC module.

Writes input data buffer bytes to the CRC data register. The configured type of transpose is applied.

Parameters

- base – CRC peripheral address.
- data – Input data stream, MSByte in data[0].
- dataSize – Size in bytes of the input data buffer.

uint32_t CRC_Get32bitResult(CRC_Type *base)

Reads the 32-bit checksum from the CRC module.

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

- base – CRC peripheral address.

Returns

An intermediate or the final 32-bit checksum, after configured transpose and complement operations.

uint16_t CRC_Get16bitResult(CRC_Type *base)

Reads a 16-bit checksum from the CRC module.

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

- base – CRC peripheral address.

Returns

An intermediate or the final 16-bit checksum, after configured transpose and complement operations.

CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT

Default configuration structure filled by *CRC_GetDefaultConfig()*. Use CRC16-CCIT-FALSE as defeault.

struct _crc_config

#include <fsl_crc.h> CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

Public Members

uint32_t polynomial

CRC Polynomial, MSBit first. Example polynomial: 0x1021 = 1_0000_0010_0001 = x^12+x^5+1

uint32_t seed

Starting checksum value

bool reflectIn

Reflect bits on input.

bool reflectOut

Reflect bits on output.

bool complementChecksum

True if the result shall be complement of the actual checksum.

crc_bits_t crcBits

Selects 16- or 32- bit CRC protocol.

crc_result_t crcResult

Selects final or intermediate checksum return from *CRC_Get16bitResult()* or *CRC_Get32bitResult()*

2.6 CTIMER: Standard counter/timers

void CTIMER_Init(CTIMER_Type *base, const *ctimer_config_t* *config)

Ungates the clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application before using the driver.

Parameters

- base – Ctimer peripheral base address
- config – Pointer to the user configuration structure.

void CTIMER_Deinit(CTIMER_Type *base)

Gates the timer clock.

Parameters

- base – Ctimer peripheral base address

void CTIMER_GetDefaultConfig(*ctimer_config_t* *config)

Fills in the timers configuration structure with the default settings.

The default values are:

```
config->mode = kCTIMER_TimerMode;
config->input = kCTIMER_Capture_0;
config->prescale = 0;
```

Parameters

- config – Pointer to the user configuration structure.

status_t CTIMER_SetupPwmPeriod(CTIMER_Type *base, const *ctimer_match_t* pwmPeriodChannel, *ctimer_match_t* matchChannel, uint32_t pwmPeriod, uint32_t pulsePeriod, bool enableInt)

Configures the PWM signal parameters.

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function can manually assign the specified channel to set the PWM cycle.

Note: When setting PWM output from multiple output pins, all should use the same PWM period

Parameters

- base – Ctimer peripheral base address
- pwmPeriodChannel – Specify the channel to control the PWM period
- matchChannel – Match pin to be used to output the PWM signal
- pwmPeriod – PWM period match value
- pulsePeriod – Pulse width match value
- enableInt – Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

Returns

kStatus_Success on success kStatus_Fail If matchChannel is equal to pwmPeriodChannel; this channel is reserved to set the PWM cycle If PWM pulse width register value is larger than 0xFFFFFFFF.

status_t CTIMER_SetupPwm(CTIMER_Type *base, const *ctimer_match_t* pwmPeriodChannel, *ctimer_match_t* matchChannel, uint8_t dutyCyclePercent, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, bool enableInt)

Configures the PWM signal parameters.

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function can manually assign the specified channel to set the PWM cycle.

Note: When setting PWM output from multiple output pins, all should use the same PWM frequency. Please use CTIMER_SetupPwmPeriod to set up the PWM with high resolution.

Parameters

- base – Ctimer peripheral base address
- pwmPeriodChannel – Specify the channel to control the PWM period
- matchChannel – Match pin to be used to output the PWM signal
- dutyCyclePercent – PWM pulse width; the value should be between 0 to 100
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – Timer counter clock in Hz
- enableInt – Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

static inline void CTIMER_UpdatePwmPulsePeriod(CTIMER_Type *base, *ctimer_match_t* matchChannel, uint32_t pulsePeriod)

Updates the pulse period of an active PWM signal.

Parameters

- base – Ctimer peripheral base address
- matchChannel – Match pin to be used to output the PWM signal
- pulsePeriod – New PWM pulse width match value

status_t CTIMER_UpdatePwmDutycycle(CTIMER_Type *base, const *ctimer_match_t* pwmPeriodChannel, *ctimer_match_t* matchChannel, uint8_t dutyCyclePercent)

Updates the duty cycle of an active PWM signal.

Note: Please use CTIMER_SetupPwmPeriod to update the PWM with high resolution. This function can manually assign the specified channel to set the PWM cycle.

Parameters

- base – Ctimer peripheral base address
- pwmPeriodChannel – Specify the channel to control the PWM period
- matchChannel – Match pin to be used to output the PWM signal

- dutyCyclePercent – New PWM pulse width; the value should be between 0 to 100

Returns

kStatus_Success on success kStatus_Fail If PWM pulse width register value is larger than 0xFFFFFFFF.

static inline void CTIMER_EnableInterrupts(CTIMER_Type *base, uint32_t mask)

Enables the selected Timer interrupts.

Parameters

- base – Ctimer peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *ctimer_interrupt_enable_t*

static inline void CTIMER_DisableInterrupts(CTIMER_Type *base, uint32_t mask)

Disables the selected Timer interrupts.

Parameters

- base – Ctimer peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *ctimer_interrupt_enable_t*

static inline uint32_t CTIMER_GetEnabledInterrupts(CTIMER_Type *base)

Gets the enabled Timer interrupts.

Parameters

- base – Ctimer peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *ctimer_interrupt_enable_t*

static inline uint32_t CTIMER_GetStatusFlags(CTIMER_Type *base)

Gets the Timer status flags.

Parameters

- base – Ctimer peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration *ctimer_status_flags_t*

static inline void CTIMER_ClearStatusFlags(CTIMER_Type *base, uint32_t mask)

Clears the Timer status flags.

Parameters

- base – Ctimer peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration *ctimer_status_flags_t*

static inline void CTIMER_StartTimer(CTIMER_Type *base)

Starts the Timer counter.

Parameters

- base – Ctimer peripheral base address

static inline void CTIMER_StopTimer(CTIMER_Type *base)

Stops the Timer counter.

Parameters

- base – Ctimer peripheral base address

FSL_CTIMER_DRIVER_VERSION

Version 2.3.6

enum _ctimer_capture_channel

List of Timer capture channels.

Values:

enumerator kCTIMER_Capture_0

Timer capture channel 0

enumerator kCTIMER_Capture_1

Timer capture channel 1

enumerator kCTIMER_Capture_3

Timer capture channel 3

enum _ctimer_capture_edge

List of capture edge options.

Values:

enumerator kCTIMER_Capture_RiseEdge

Capture on rising edge

enumerator kCTIMER_Capture_FallEdge

Capture on falling edge

enumerator kCTIMER_Capture_BothEdge

Capture on rising and falling edge

enum _ctimer_match

List of Timer match registers.

Values:

enumerator kCTIMER_Match_0

Timer match register 0

enumerator kCTIMER_Match_1

Timer match register 1

enumerator kCTIMER_Match_2

Timer match register 2

enumerator kCTIMER_Match_3
Timer match register 3

enum _ctimer_external_match
List of external match.
Values:

enumerator kCTIMER_External_Match_0
External match 0

enumerator kCTIMER_External_Match_1
External match 1

enumerator kCTIMER_External_Match_2
External match 2

enumerator kCTIMER_External_Match_3
External match 3

enum _ctimer_match_output_control
List of output control options.
Values:

enumerator kCTIMER_Output_NoAction
No action is taken

enumerator kCTIMER_Output_Clear
Clear the EM bit/output to 0

enumerator kCTIMER_Output_Set
Set the EM bit/output to 1

enumerator kCTIMER_Output_Toggle
Toggle the EM bit/output

enum _ctimer_timer_mode
List of Timer modes.
Values:

enumerator kCTIMER_TimerMode

enumerator kCTIMER_IncreaseOnRiseEdge

enumerator kCTIMER_IncreaseOnFallEdge

enumerator kCTIMER_IncreaseOnBothEdge

enum _ctimer_interrupt_enable

List of Timer interrupts.

Values:

enumerator kCTIMER_Match0InterruptEnable

Match 0 interrupt

enumerator kCTIMER_Match1InterruptEnable

Match 1 interrupt

enumerator kCTIMER_Match2InterruptEnable

Match 2 interrupt

enumerator kCTIMER_Match3InterruptEnable

Match 3 interrupt

enum _ctimer_status_flags

List of Timer flags.

Values:

enumerator kCTIMER_Match0Flag

Match 0 interrupt flag

enumerator kCTIMER_Match1Flag

Match 1 interrupt flag

enumerator kCTIMER_Match2Flag

Match 2 interrupt flag

enumerator kCTIMER_Match3Flag

Match 3 interrupt flag

enum ctimer_callback_type_t

Callback type when registering for a callback. When registering a callback an array of function pointers is passed the size could be 1 or 8, the callback type will tell that.

Values:

enumerator kCTIMER_SingleCallback

Single Callback type where there is only one callback for the timer. based on the status flags different channels needs to be handled differently

enumerator kCTIMER_MultipleCallback

Multiple Callback type where there can be 8 valid callbacks, one per channel. for both match/capture

typedef enum *_ctimer_capture_channel* ctimer_capture_channel_t

List of Timer capture channels.

typedef enum *_ctimer_capture_edge* ctimer_capture_edge_t

List of capture edge options.

typedef enum *_ctimer_match* ctimer_match_t

List of Timer match registers.

typedef enum *_ctimer_external_match* ctimer_external_match_t

List of external match.

typedef enum *_ctimer_match_output_control* ctimer_match_output_control_t

List of output control options.

typedef enum *_ctimer_timer_mode* ctimer_timer_mode_t

List of Timer modes.

typedef enum *_ctimer_interrupt_enable* ctimer_interrupt_enable_t

List of Timer interrupts.

typedef enum *_ctimer_status_flags* ctimer_status_flags_t

List of Timer flags.

typedef void (*ctimer_callback_t)(uint32_t flags)

typedef struct *_ctimer_match_config* ctimer_match_config_t

Match configuration.

This structure holds the configuration settings for each match register.

typedef struct *_ctimer_config* ctimer_config_t

Timer configuration structure.

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the *CTIMER_GetDefaultConfig()* function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

void CTIMER_SetupMatch(CTIMER_Type *base, *ctimer_match_t* matchChannel, const *ctimer_match_config_t* *config)

Setup the match register.

User configuration is used to setup the match value and action to be taken when a match occurs.

Parameters

- base – Ctimer peripheral base address
- matchChannel – Match register to configure
- config – Pointer to the match configuration structure

uint32_t CTIMER_GetOutputMatchStatus(CTIMER_Type *base, uint32_t matchChannel)

Get the status of output match.

This function gets the status of output MAT, whether or not this output is connected to a pin. This status is driven to the MAT pins if the match function is selected via IOCON. 0 = LOW. 1 = HIGH.

Parameters

- base – Ctimer peripheral base address
- matchChannel – External match channel, user can obtain the status of multiple match channels at the same time by using the logic of "|" enumeration *ctimer_external_match_t*

Returns

The mask of external match channel status flags. Users need to use the _ctimer_external_match type to decode the return variables.

void CTIMER_SetupCapture(CTIMER_Type *base, *ctimer_capture_channel_t* capture, *ctimer_capture_edge_t* edge, bool enableInt)

Setup the capture.

Parameters

- base – Ctimer peripheral base address
- capture – Capture channel to configure
- edge – Edge on the channel that will trigger a capture
- enableInt – Flag to enable channel interrupts, if enabled then the registered call back is called upon capture

static inline uint32_t CTIMER_GetTimerCountValue(CTIMER_Type *base)

Get the timer count value from TC register.

Parameters

- base – Ctimer peripheral base address.

Returns

return the timer count value.

void CTIMER_RegisterCallBack(CTIMER_Type *base, *ctimer_callback_t* *cb_func, *ctimer_callback_type_t* cb_type)

Register callback.

This function configures CTimer Callback in following modes:

- Single Callback: cb_func should be pointer to callback function pointer For example: ctimer_callback_t ctimer_callback = pwm_match_callback; CTIMER_RegisterCallBack(CTIMER, &ctimer_callback, kCTIMER_SingleCallback);
- Multiple Callback: cb_func should be pointer to array of callback function pointers Each element corresponds to Interrupt Flag in IR register. For example: ctimer_callback_t ctimer_callback_table[] = { ctimer_match0_callback, NULL, NULL, ctimer_match3_callback, NULL, NULL, NULL, NULL}; CTIMER_RegisterCallBack(CTIMER, &ctimer_callback_table[0], kCTIMER_MultipleCallback);

Parameters

- base – Ctimer peripheral base address
- cb_func – Pointer to callback function pointer
- cb_type – callback function type, singular or multiple

static inline void CTIMER_Reset(CTIMER_Type *base)

Reset the counter.

The timer counter and prescale counter are reset on the next positive edge of the APB clock.

Parameters

- base – Ctimer peripheral base address

static inline void CTIMER_SetPrescale(CTIMER_Type *base, uint32_t prescale)

Setup the timer prescale value.

Specifies the maximum value for the Prescale Counter.

Parameters

- base – Ctimer peripheral base address
- prescale – Prescale value

static inline uint32_t CTIMER_GetCaptureValue(CTIMER_Type *base, *ctimer_capture_channel_t* capture)

Get capture channel value.

Get the counter/timer value on the corresponding capture channel.

Parameters

- base – Ctimer peripheral base address
- capture – Select capture channel

Returns

The timer count capture value.

static inline void CTIMER_EnableResetMatchChannel(CTIMER_Type *base, *ctimer_match_t* match, bool enable)

Enable reset match channel.

Set the specified match channel reset operation.

Parameters

- base – Ctimer peripheral base address
- match – match channel used
- enable – Enable match channel reset operation.

static inline void CTIMER_EnableStopMatchChannel(CTIMER_Type *base, *ctimer_match_t* match, bool enable)

Enable stop match channel.

Set the specified match channel stop operation.

Parameters

- base – Ctimer peripheral base address.
- match – match channel used.
- enable – Enable match channel stop operation.

static inline void CTIMER_EnableMatchChannelReload(CTIMER_Type *base, *ctimer_match_t* match, bool enable)

Enable reload channel falling edge.

Enable the specified match channel reload match shadow value.

Parameters

- base – Ctimer peripheral base address.
- match – match channel used.
- enable – Enable .

static inline void CTIMER_EnableRisingEdgeCapture(CTIMER_Type *base, *ctimer_capture_channel_t* capture, bool enable)

Enable capture channel rising edge.

Sets the specified capture channel for rising edge capture.

Parameters

- base – Ctimer peripheral base address.
- capture – capture channel used.
- enable – Enable rising edge capture.

static inline void CTIMER_EnableFallingEdgeCapture(CTIMER_Type *base, *ctimer_capture_channel_t* capture, bool enable)

Enable capture channel falling edge.

Sets the specified capture channel for falling edge capture.

Parameters

- base – Ctimer peripheral base address.
- capture – capture channel used.
- enable – Enable falling edge capture.

static inline void CTIMER_SetShadowValue(CTIMER_Type *base, *ctimer_match_t* match, uint32_t matchvalue)

Set the specified match shadow channel.

Parameters

- base – Ctimer peripheral base address.
- match – match channel used.
- matchvalue – Reload the value of the corresponding match register.

struct __ctimer_match_config

#include <fsl_ctimer.h> Match configuration.

This structure holds the configuration settings for each match register.

Public Members

uint32_t matchValue

This is stored in the match register

bool enableCounterReset

true: Match will reset the counter false: Match will not reser the counter

bool enableCounterStop

true: Match will stop the counter false: Match will not stop the counter

ctimer_match_output_control_t outControl

Action to be taken on a match on the EM bit/output

bool outPinInitState

Initial value of the EM bit/output

bool enableInterrupt

true: Generate interrupt upon match false: Do not generate interrupt on match

struct _ctimer_config

#include <fsl_ctimer.h> Timer configuration structure.

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the *CTIMER_GetDefaultConfig()* function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Public Members

ctimer_timer_mode_t mode

Timer mode

ctimer_capture_channel_t input

Input channel to increment the timer, used only in timer modes that rely on this input signal to increment TC

uint32_t prescale

Prescale value

2.7 Dsp

enum _dsp_static_vec_sel

Fusion DSP vector table select.

Values:

enumerator kDSP_StatVecSel0

enumerator kDSP_StatVecSel1

typedef struct *_dsp_copy_image* dsp_copy_image_t

Structure for DSP copy image to destination address.

Defines start and destination address for copying image with given size.

typedef enum *_dsp_static_vec_sel* dsp_static_vec_sel_t

Fusion DSP vector table select.

void DSP_Init(void)

Initializing DSP core.

Power up DSP TCM Enable DSP clock Reset DSP peripheral

void DSP_Deinit(void)

Deinit DSP core.

Power down DSP TCM Disable DSP clock Set DSP peripheral reset

void DSP_CopyImage(*dsp_copy_image_t* *dspCopyImage)

Copy DSP image to destination address.

Copy DSP image from source address to destination address with given size.

Parameters

- dspCopyImage – Structure contains information for DSP copy image to destination address.

static inline void DSP_Start(void)

Start DSP core.

static inline void DSP_Stop(void)

Stop DSP core.

static inline void DSP_SetVecRemap(*dsp_static_vec_sel_t* statVecSel, uint32_t remap)

Set DSP static vector table remap.

Parameters

- statVecSel – static vector base address selection
- remap – static vector remap, For HiFi1 when statVecSel = 0, the valid range is 0~0x3FF, when statVecSel=1, the valid range is 0~0x7FF. For HiFi4, the bits value will remap ARADDR[22:10] when [STATVECSELECT] is high and ARADDR hits 1 KB from 2010_0000h.

FSL_DSP_DRIVER_VERSION

DSP driver version 2.0.0.

uint32_t *srcAddr

uint32_t *destAddr

uint32_t size

struct _dsp_copy_image

#include <fsl_dsp.h> Structure for DSP copy image to destination address.

Defines start and destination address for copying image with given size.

2.8 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

void EDMA_Init(*EDMA_Type* *base, const *edma_config_t* *config)

Initializes the eDMA peripheral.

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure. All emda enabled request will be cleared in this function.

Note: This function enables the minor loop map feature.

Parameters

- base – eDMA peripheral base address.
- config – A pointer to the configuration structure, see "edma_config_t".

void EDMA_Deinit(*EDMA_Type* *base)

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

Parameters

- base – eDMA peripheral base address.

void EDMA_InstallTCD(*EDMA_Type* *base, uint32_t channel, *edma_tcd_t* *tcd)

Push content of TCD structure into hardware TCD register.

Parameters

- base – EDMA peripheral base address.
- channel – EDMA channel number.
- tcd – Point to TCD structure.

void EDMA_GetDefaultConfig(*edma_config_t* *config)

Gets the eDMA default configuration structure.

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
config.enableContinuousLinkMode = false;
config.enableHaltOnError = true;
config.enableRoundRobinArbitration = false;
config.enableDebugMode = false;
```

Parameters

- config – A pointer to the eDMA configuration structure.

void EDMA_InitChannel(*EDMA_Type* *base, uint32_t channel, *edma_channel_config_t* *channelConfig)

EDMA Channel initialization.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- channelConfig – pointer to user's eDMA4 channel config structure, see edma_channel_config_t for detail.

static inline void EDMA_SetChannelMemoryAttribute(*EDMA_Type* *base, uint32_t channel, *edma_channel_memory_attribute_t* writeAttribute, *edma_channel_memory_attribute_t* readAttribute)

Set channel memory attribute.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- writeAttribute – Attributes associated with a write transaction.

- readAttribute – Attributes associated with a read transaction.

static inline void EDMA_SetChannelSignExtension(*EDMA_Type* *base, uint32_t channel, uint8_t position)

Set channel sign extension.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- position – A non-zero value specifing the sign extend bit position. If 0, sign extension is disabled.

static inline void EDMA_SetChannelSwapSize(*EDMA_Type* *base, uint32_t channel, *edma_channel_swap_size_t* swapSize)

Set channel swap size.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- swapSize – Swap occurs with respect to the specified transfer size. If 0, swap is disabled.

static inline void EDMA_SetChannelAccessType(*EDMA_Type* *base, uint32_t channel, *edma_channel_access_type_t* channelAccessType)

Set channel access type.

Parameters

- base – eDMA4 peripheral base address.
- channel – eDMA4 channel number.
- channelAccessType – eDMA4's transactions type on the system bus when the channel is active.

static inline void EDMA_SetChannelMux(*EDMA_Type* *base, uint32_t channel, uint32_t channelRequestSource)

Set channel request source.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- channelRequestSource – eDMA hardware service request source for the channel. User need to use the dma_request_source_t type as the input parameter. Note that devices may use other enum type to express dma request source and User can fined it in SOC header or fsl_edma_soc.h.

static inline uint32_t EDMA_GetChannelSystemBusInformation(*EDMA_Type* *base, uint32_t channel)

Gets the channel identification and attribute information on the system bus interface.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

The mask of the channel system bus information. Users need to use the _edma_channel_sys_bus_info type to decode the return variables.

static inline void EDMA_EnableChannelMasterIDReplication(*EDMA_Type* *base, uint32_t channel, bool enable)

Set channel master ID replication.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – true is enable, false is disable.

static inline void EDMA_SetChannelProtectionLevel(*EDMA_Type* *base, uint32_t channel, *edma_channel_protection_level_t* level)

Set channel security level.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- level – security level.

void EDMA_ResetChannel(*EDMA_Type* *base, uint32_t channel)

Sets all TCD registers to default values.

This function sets TCD registers for this channel to default values.

Note: This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

Note: This function enables the auto stop request feature.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

void EDMA_SetTransferConfig(*EDMA_Type* *base, uint32_t channel, const *edma_transfer_config_t* *config, *edma_tcd_t* *nextTcd)

Configures the eDMA transfer attribute.

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address. Example:

```
edma_transfer_t config;
edma_tcd_t tcd;
config.srcAddr = ..;
config.destAddr = ..;
...
EDMA_SetTransferConfig(DMA0, channel, &config, &stcd);
```

Note: If nextTcd is not NULL, it means scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the eDMA_ResetChannel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Point to TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

void EDMA_SetMinorOffsetConfig(*EDMA_Type* *base, uint32_t channel, const *edma_minor_offset_config_t* *config)

Configures the eDMA minor offset feature.

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – A pointer to the minor offset configuration structure.

void EDMA_SetChannelPreemptionConfig(*EDMA_Type* *base, uint32_t channel, const *edma_channel_Preemption_config_t* *config)

Configures the eDMA channel preemption feature.

This function configures the channel preemption attribute and the priority of the channel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number
- config – A pointer to the channel preemption configuration structure.

void EDMA_SetChannelLink(*EDMA_Type* *base, uint32_t channel, *edma_channel_link_type_t* type, uint32_t linkedChannel)

Sets the channel link for the eDMA transfer.

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- type – A channel link type, which can be one of the following:
  - kEDMA_LinkNone
  - kEDMA_MinorLink

- kEDMA_MajorLink

- linkedChannel – The linked channel number.

void EDMA_SetBandWidth(*EDMA_Type* *base, uint32_t channel, *edma_bandwidth_t* bandWidth)

Sets the bandwidth for the eDMA transfer.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- bandWidth – A bandwidth setting, which can be one of the following:
  - kEDMABandwidthStallNone
  - kEDMABandwidthStall4Cycle
  - kEDMABandwidthStall8Cycle

void EDMA_SetModulo(*EDMA_Type* *base, uint32_t channel, *edma_modulo_t* srcModulo, *edma_modulo_t* destModulo)

Sets the source modulo and the destination modulo for the eDMA transfer.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

static inline void EDMA_EnableAsyncRequest(*EDMA_Type* *base, uint32_t channel, bool enable)

Enables an async request for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

static inline void EDMA_EnableAutoStopRequest(*EDMA_Type* *base, uint32_t channel, bool enable)

Enables an auto stop request for the eDMA transfer.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

void EDMA_EnableChannelInterrupts(*EDMA_Type* *base, uint32_t channel, uint32_t mask)

Enables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

void EDMA_DisableChannelInterrupts(*EDMA_Type* *base, uint32_t channel, uint32_t mask)

Disables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of the interrupt source to be set. Use the defined edma_interrupt_enable_t type.

void EDMA_SetMajorOffsetConfig(*EDMA_Type* *base, uint32_t channel, int32_t sourceOffset, int32_t destOffset)

Configures the eDMA channel TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- base – eDMA peripheral base address.
- channel – edma channel number.
- sourceOffset – source address offset will be applied to source address after major loop done.
- destOffset – destination address offset will be applied to source address after major loop done.

void EDMA_ConfigChannelSoftwareTCD(*edma_tcd_t* *tcd, const *edma_transfer_config_t* *transfer)

Sets TCD fields according to the user's channel transfer configuration structure, *edma_transfer_config_t*.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA_ConfigChannelSoftwareTCDExt*

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

Note: This function enables the auto stop request feature.

Parameters

- tcd – Pointer to the TCD structure.
- transfer – channel transfer configuration pointer.

void EDMA_TcdReset(*edma_tcd_t* *tcd)

Sets all fields to default values for the TCD structure.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA_TcdResetExt*

This function sets all fields for this TCD structure to default value.

Note: This function enables the auto stop request feature.

Parameters

- tcd – Pointer to the TCD structure.

void EDMA_TcdSetTransferConfig(*edma_tcd_t* *tcd, const *edma_transfer_config_t* *config, *edma_tcd_t* *nextTcd)

Configures the eDMA TCD transfer attribute.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA_TcdSetTransferConfigExt*

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_t config = {
...
}
edma_tcd_t tcd __aligned(32);
edma_tcd_t nextTcd __aligned(32);
EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

Note: TCD address should be 32 bytes aligned or it causes an eDMA error.

Note: If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA_TcdReset.

Parameters

- tcd – Pointer to the TCD structure.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

void EDMA_TcdSetMinorOffsetConfig(*edma_tcd_t* *tcd, const *edma_minor_offset_config_t* *config)

Configures the eDMA TCD minor offset feature.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA_TcdSetMinorOffsetConfigExt*

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

- tcd – A point to the TCD structure.
- config – A pointer to the minor offset configuration structure.

void EDMA_TcdSetChannelLink(*edma_tcd_t* *tcd, *edma_channel_link_type_t* type, uint32_t linkedChannel)

Sets the channel link for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA_TcdSetChannelLinkExt*

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- tcd – Point to the TCD structure.
- type – Channel link type, it can be one of:
  - kEDMA_LinkNone
  - kEDMA_MinorLink
  - kEDMA_MajorLink
- linkedChannel – The linked channel number.

static inline void EDMA_TcdSetBandWidth(*edma_tcd_t* *tcd, *edma_bandwidth_t* bandWidth)

Sets the bandwidth for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA_TcdSetBandWidthExt*

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- tcd – A pointer to the TCD structure.
- bandWidth – A bandwidth setting, which can be one of the following:
  - kEDMABandwidthStallNone
  - kEDMABandwidthStall4Cycle
  - kEDMABandwidthStall8Cycle

void EDMA_TcdSetModulo(*edma_tcd_t* *tcd, *edma_modulo_t* srcModulo, *edma_modulo_t* destModulo)

Sets the source modulo and the destination modulo for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA_TcdSetModuloExt*

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- tcd – A pointer to the TCD structure.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

static inline void EDMA_TcdEnableAutoStopRequest(*edma_tcd_t* *tcd, bool enable)

Sets the auto stop request for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA_TcdEnableAutoStopRequestExt*

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- tcd – A pointer to the TCD structure.
- enable – The command to enable (true) or disable (false).

void EDMA_TcdEnableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Enables the interrupt source for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA_TcdEnableInterruptsExt*

Parameters

- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

void EDMA_TcdDisableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Disables the interrupt source for the eDMA TCD.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA_TcdDisableInterruptsExt*

Parameters

- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

void EDMA_TcdSetMajorOffsetConfig(*edma_tcd_t* *tcd, int32_t sourceOffset, int32_t destOffset)

Configures the eDMA TCD major offset feature.

@Note This API only supports EDMA4 TCD type. It can be used to support all types with extension API *EDMA_TcdSetMajorOffsetConfigExt*

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- tcd – A point to the TCD structure.
- sourceOffset – source address offset wiil be applied to source address after major loop done.
- destOffset – destination address offset will be applied to source address after major loop done.

void EDMA_ConfigChannelSoftwareTCDExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, const *edma_transfer_config_t* *transfer)

Sets TCD fields according to the user's channel transfer configuration structure, *edma_transfer_config_t*.

Application should be careful about the TCD pool buffer storage class,

- For the platform has cache, the software TCD should be put in non cache section
- The TCD pool buffer should have a consistent storage class.

Note: This function enables the auto stop request feature.

Parameters

- base – eDMA peripheral base address.
- tcd – Pointer to the TCD structure.
- transfer – channel transfer configuration pointer.

void EDMA_TcdResetExt(*EDMA_Type* *base, *edma_tcd_t* *tcd)

Sets all fields to default values for the TCD structure.

This function sets all fields for this TCD structure to default value.

Note: This function enables the auto stop request feature.

Parameters

- base – eDMA peripheral base address.
- tcd – Pointer to the TCD structure.

void EDMA_TcdSetTransferConfigExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, const *edma_transfer_config_t* *config, *edma_tcd_t* *nextTcd)

Configures the eDMA TCD transfer attribute.

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_t config = {
...
}
edma_tcd_t tcd __aligned(32);
edma_tcd_t nextTcd __aligned(32);
EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

Note: TCD address should be 32 bytes aligned or it causes an eDMA error.

Note: If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA_TcdReset.

Parameters

- base – eDMA peripheral base address.
- tcd – Pointer to the TCD structure.
- config – Pointer to eDMA transfer configuration structure.

- nextTcd – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

void EDMA_TcdSetMinorOffsetConfigExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, const *edma_minor_offset_config_t* *config)

Configures the eDMA TCD minor offset feature.

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

- base – eDMA peripheral base address.
- tcd – A point to the TCD structure.
- config – A pointer to the minor offset configuration structure.

void EDMA_TcdSetChannelLinkExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, *edma_channel_link_type_t* type, uint32_t linkedChannel)

Sets the channel link for the eDMA TCD.

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- type – Channel link type, it can be one of:
  - kEDMA_LinkNone
  - kEDMA_MinorLink
  - kEDMA_MajorLink
- linkedChannel – The linked channel number.

static inline void EDMA_TcdSetBandWidthExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, *edma_bandwidth_t* bandWidth)

Sets the bandwidth for the eDMA TCD.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- bandWidth – A bandwidth setting, which can be one of the following:
  - kEDMABandwidthStallNone
  - kEDMABandwidthStall4Cycle
  - kEDMABandwidthStall8Cycle

void EDMA_TcdSetModuloExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, *edma_modulo_t* srcModulo, *edma_modulo_t* destModulo)

Sets the source modulo and the destination modulo for the eDMA TCD.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

static inline void EDMA_TcdEnableAutoStopRequestExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, bool enable)

Sets the auto stop request for the eDMA TCD.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- tcd – A pointer to the TCD structure.
- enable – The command to enable (true) or disable (false).

void EDMA_TcdEnableInterruptsExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, uint32_t mask)

Enables the interrupt source for the eDMA TCD.

Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

void EDMA_TcdDisableInterruptsExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, uint32_t mask)

Disables the interrupt source for the eDMA TCD.

Parameters

- base – eDMA peripheral base address.
- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

void EDMA_TcdSetMajorOffsetConfigExt(*EDMA_Type* *base, *edma_tcd_t* *tcd, int32_t sourceOffset, int32_t destOffset)

Configures the eDMA TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- base – eDMA peripheral base address.
- tcd – A point to the TCD structure.
- sourceOffset – source address offset wiil be applied to source address after major loop done.

- destOffset – destination address offset will be applied to source address after major loop done.

static inline void EDMA_EnableChannelRequest(*EDMA_Type* *base, uint32_t channel)

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

static inline void EDMA_DisableChannelRequest(*EDMA_Type* *base, uint32_t channel)

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

static inline void EDMA_TriggerChannelStart(*EDMA_Type* *base, uint32_t channel)

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

uint32_t EDMA_GetRemainingMajorLoopCount(*EDMA_Type* *base, uint32_t channel)

Gets the remaining major loop count from the eDMA current channel TCD.

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

Note: 1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.

a. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA_TCDn_NBYTES_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: RemainingBytes = RemainingMajorLoopCount * NBYTES(initially configured)

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

Major loop count which has not been transferred yet for the current TCD.

static inline uint32_t EDMA_GetErrorStatusFlags(*EDMA_Type* *base)

Gets the eDMA channel error status flags.

Parameters

- base – eDMA peripheral base address.

Returns

The mask of error status flags. Users need to use the _edma_error_status_flags type to decode the return variables.

uint32_t EDMA_GetChannelStatusFlags(*EDMA_Type* *base, uint32_t channel)

Gets the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.

Returns

The mask of channel status flags. Users need to use the _edma_channel_status_flags type to decode the return variables.

void EDMA_ClearChannelStatusFlags(*EDMA_Type* *base, uint32_t channel, uint32_t mask)

Clears the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of channel status to be cleared. Users need to use the defined _edma_channel_status_flags type.

status_t EDMA_CreateHandle(*edma_handle_t* *handle, *EDMA_Type* *base, uint32_t channel)

Creates the eDMA handle.

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

- handle – eDMA handle pointer. The eDMA handle stores callback function and parameters.
- base – eDMA peripheral base address.
- channel – eDMA channel number.

Return values

- kStatus_Success –
- kStatus_InvalidArgument –

void EDMA_InstallTCDMemory(*edma_handle_t* *handle, *edma_tcd_t* *tcdPool, uint32_t tcdSize)

Installs the TCDs memory pool into the eDMA handle.

This function is called after the EDMA_CreateHandle to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a new transfer. Users need to prepare tcd memory and also configure tcds using interface EDMA_SubmitTransfer.

Parameters

- handle – eDMA handle pointer.

- tcdPool – A memory pool to store TCDs. It must be 32 bytes aligned.
- tcdSize – The number of TCD slots.

void EDMA_SetCallback(*edma_handle_t* *handle, *edma_callback* callback, void *userData)

Installs a callback function for the eDMA transfer.

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

Parameters

- handle – eDMA handle pointer.
- callback – eDMA callback function pointer.
- userData – A parameter for the callback function.

void EDMA_PrepareTransferConfig(*edma_transfer_config_t* *config, void *srcAddr, uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes)

Prepares the eDMA transfer structure configurations.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE). User can check if 128 bytes support is available for specific instance by FSL_FEATURE_EDMA_INSTANCE_SUPPORT_128_BYTES_TRANSFERn.

Parameters

- config – The user configuration structure of type edma_transfer_t.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- srcOffset – source address offset.
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- destOffset – destination address offset.
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.

void EDMA_PrepareTransfer(*edma_transfer_config_t* *config, void *srcAddr, uint32_t srcWidth, void *destAddr, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferBytes, *edma_transfer_type_t* type)

Prepares the eDMA transfer structure.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- config – The user configuration structure of type edma_transfer_t.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.
- type – eDMA transfer type.

void EDMA_PrepareTransferTCD(*edma_handle_t* *handle, *edma_tcd_t* *tcd, void *srcAddr, uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes, *edma_tcd_t* *nextTcd)

Prepares the eDMA transfer content descriptor.

This function prepares the transfer content descriptor structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- handle – eDMA handle pointer.
- tcd – Pointer to eDMA transfer content descriptor structure.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- srcOffset – source address offset.
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- destOffset – destination address offset.
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.
- nextTcd – eDMA transfer linked TCD address.

status_t EDMA_SubmitTransferTCD(*edma_handle_t* *handle, *edma_tcd_t* *tcd)

Submits the eDMA transfer content descriptor.

This function submits the eDMA transfer request according to the transfer content descriptor. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA_InstallTCDMemory before.

Typical user case:

a. submit single transfer

```
edma_tcd_t tcd;
EDMA_PrepareTransferTCD(handle, tcd, ....)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_StartTransfer(handle)
```

b. submit static link transfer,

```
edma_tcd_t tcd[2];
EDMA_PrepareTransferTCD(handle, &tcd[0], ....)
EDMA_PrepareTransferTCD(handle, &tcd[1], ....)
EDMA_SubmitTransferTCD(handle, &tcd[0])
EDMA_StartTransfer(handle)
```

c. submit dynamic link transfer

```
edma_tcd_t tcdpool[2];
EDMA_InstallTCDMemory(&g_DMA_Handle, tcdpool, 2);
edma_tcd_t tcd;
EDMA_PrepareTransferTCD(handle, tcd, ....)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_PrepareTransferTCD(handle, tcd, ....)
EDMA_SubmitTransferTCD(handle, tcd)
EDMA_StartTransfer(handle)
```

d. submit loop transfer

```
edma_tcd_t tcd[2];
EDMA_PrepareTransferTCD(handle, &tcd[0], ...,&tcd[1])
EDMA_PrepareTransferTCD(handle, &tcd[1], ..., &tcd[0])
EDMA_SubmitTransferTCD(handle, &tcd[0])
EDMA_StartTransfer(handle)
```

Parameters

- handle – eDMA handle pointer.
- tcd – Pointer to eDMA transfer content descriptor structure.

Return values

- kStatus_EDMA_Success – It means submit transfer request succeed.
- kStatus_EDMA_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus_EDMA_Busy – It means the given channel is busy, need to submit request later.

status_t EDMA_SubmitTransfer(*edma_handle_t* *handle, const *edma_transfer_config_t* *config)

Submits the eDMA transfer request.

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA_InstallTCDMemory before.

Parameters

- handle – eDMA handle pointer.
- config – Pointer to eDMA transfer configuration structure.

Return values

- kStatus_EDMA_Success – It means submit transfer request succeed.
- kStatus_EDMA_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus_EDMA_Busy – It means the given channel is busy, need to submit request later.

status_t EDMA_SubmitLoopTransfer(*edma_handle_t* *handle, *edma_transfer_config_t* *transfer, uint32_t transferLoopCount)

Submits the eDMA scatter gather transfer configurations.

The function is target for submit loop transfer request, the ring transfer request means that the transfer request TAIL is link to HEAD, such as, A->B->C->D->A, or A->A

To use the ring transfer feature, the application should allocate several transfer object, such as

```
edma_channel_transfer_config_t transfer[2];
EDMA_TransferSubmitLoopTransfer(psHandle, &transfer, 2U);
```

Then eDMA driver will link transfer[0] and transfer[1] to each other

---

Note: Application should check the return value of this function to avoid transfer request submit failed

---

Parameters

- handle – eDMA handle pointer
- transfer – pointer to user's eDMA channel configure structure, see edma_channel_transfer_config_t for detail
- transferLoopCount – the count of the transfer ring, if loop count is 1, that means that the one will link to itself.

Return values

- kStatus_Success – It means submit transfer request succeed
- kStatus_EDMA_Busy – channel is in busy status
- kStatus_InvalidArgument – Invalid Argument

void EDMA_StartTransfer(*edma_handle_t* *handle)

eDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

- handle – eDMA handle pointer.

void EDMA_StopTransfer(*edma_handle_t* *handle)

eDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call *EDMA_StartTransfer()* again to resume the transfer.

Parameters

- handle – eDMA handle pointer.

void EDMA_AbortTransfer(*edma_handle_t* *handle)

eDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

- handle – DMA handle pointer.

static inline uint32_t EDMA_GetUnusedTCDNumber(*edma_handle_t* *handle)

Get unused TCD slot number.

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

- handle – DMA handle pointer.

Returns

The unused tcd slot number.

static inline uint32_t EDMA_GetNextTCDAddress(*edma_handle_t* *handle)

Get the next tcd address.

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

- handle – DMA handle pointer.

Returns

The next TCD address.

void EDMA_HandleIRQ(*edma_handle_t* *handle)

eDMA IRQ handler for the current major loop transfer completion.

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga_index are calculated based on the DLAST_SGA bitfield lies in the TCD_CSR register, the sga_index in this case should be 2 (DLAST_SGA of TCD[1] stores the address of TCD[2]). Thus, the "tcdUsed" updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the "eDMA basic data flow" in the eDMA Functional description section of the Reference Manual for further details.

Parameters

- handle – eDMA handle pointer.

void EDMA_TcdInit(*EDMA_Type* *base, *edma_tcd_t* *tcdRegs)

Initialize all fields to 0 for the TCD structure.

This function initialize all fields for this TCD structure to 0.

Parameters

- base – eDMA peripheral base address.
- tcd – Pointer to the TCD structure.

FSL_EDMA_DRIVER_VERSION

eDMA driver version

Version 2.10.14.

_edma_transfer_status eDMA transfer status

Values:

enumerator kStatus_EDMA_QueueFull

TCD queue is full.

enumerator kStatus_EDMA_Busy

Channel is busy and can't handle the transfer request.

enum _edma_transfer_size

eDMA transfer configuration

Values:

enumerator kEDMA_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kEDMA_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA_TransferSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA_TransferSize16Bytes

Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA_TransferSize32Bytes

Source/Destination data transfer size is 32 bytes every time

enumerator kEDMA_TransferSize64Bytes

Source/Destination data transfer size is 64 bytes every time

enumerator kEDMA_TransferSize128Bytes

Source/Destination data transfer size is 128 bytes every time

enum _edma_modulo

eDMA modulo configuration

Values:

enumerator kEDMA_ModuloDisable

Disable modulo

enumerator kEDMA_Modulo2bytes
Circular buffer size is 2 bytes.

enumerator kEDMA_Modulo4bytes
Circular buffer size is 4 bytes.

enumerator kEDMA_Modulo8bytes
Circular buffer size is 8 bytes.

enumerator kEDMA_Modulo16bytes
Circular buffer size is 16 bytes.

enumerator kEDMA_Modulo32bytes
Circular buffer size is 32 bytes.

enumerator kEDMA_Modulo64bytes
Circular buffer size is 64 bytes.

enumerator kEDMA_Modulo128bytes
Circular buffer size is 128 bytes.

enumerator kEDMA_Modulo256bytes
Circular buffer size is 256 bytes.

enumerator kEDMA_Modulo512bytes
Circular buffer size is 512 bytes.

enumerator kEDMA_Modulo1Kbytes
Circular buffer size is 1 K bytes.

enumerator kEDMA_Modulo2Kbytes
Circular buffer size is 2 K bytes.

enumerator kEDMA_Modulo4Kbytes
Circular buffer size is 4 K bytes.

enumerator kEDMA_Modulo8Kbytes
Circular buffer size is 8 K bytes.

enumerator kEDMA_Modulo16Kbytes
Circular buffer size is 16 K bytes.

enumerator kEDMA_Modulo32Kbytes
Circular buffer size is 32 K bytes.

enumerator kEDMA_Modulo64Kbytes
Circular buffer size is 64 K bytes.

enumerator kEDMA_Modulo128Kbytes
Circular buffer size is 128 K bytes.

enumerator kEDMA_Modulo256Kbytes
Circular buffer size is 256 K bytes.

enumerator kEDMA_Modulo512Kbytes
Circular buffer size is 512 K bytes.

enumerator kEDMA_Modulo1Mbytes
Circular buffer size is 1 M bytes.

enumerator kEDMA_Modulo2Mbytes
Circular buffer size is 2 M bytes.

enumerator kEDMA_Modulo4Mbytes
Circular buffer size is 4 M bytes.

enumerator kEDMA_Modulo8Mbytes
Circular buffer size is 8 M bytes.

enumerator kEDMA_Modulo16Mbytes
Circular buffer size is 16 M bytes.

enumerator kEDMA_Modulo32Mbytes
Circular buffer size is 32 M bytes.

enumerator kEDMA_Modulo64Mbytes
Circular buffer size is 64 M bytes.

enumerator kEDMA_Modulo128Mbytes
Circular buffer size is 128 M bytes.

enumerator kEDMA_Modulo256Mbytes
Circular buffer size is 256 M bytes.

enumerator kEDMA_Modulo512Mbytes
Circular buffer size is 512 M bytes.

enumerator kEDMA_Modulo1Gbytes
Circular buffer size is 1 G bytes.

enumerator kEDMA_Modulo2Gbytes
Circular buffer size is 2 G bytes.

enum _edma_bandwidth
Bandwidth control.

Values:

enumerator kEDMA_BandwidthStallNone
No eDMA engine stalls.

enumerator kEDMA_BandwidthStall4Cycle
eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA_BandwidthStall8Cycle
eDMA engine stalls for 8 cycles after each read/write.

enum _edma_channel_link_type
Channel link type.
Values:

enumerator kEDMA_LinkNone
No channel link

enumerator kEDMA_MinorLink
Channel link after each minor loop

enumerator kEDMA_MajorLink
Channel link while major loop count exhausted

_edma_channel_status_flags eDMA channel status flags.
Values:

enumerator kEDMA_DoneFlag
DONE flag, set while transfer finished, CITER value exhausted

enumerator kEDMA_ErrorFlag
eDMA error flag, an error occurred in a transfer

enumerator kEDMA_InterruptFlag
eDMA interrupt flag, set while an interrupt occurred of this channel

_edma_error_status_flags eDMA channel error status flags.
Values:

enumerator kEDMA_DestinationBusErrorFlag
Bus error on destination address

enumerator kEDMA_SourceBusErrorFlag
Bus error on the source address

enumerator kEDMA_ScatterGatherErrorFlag
Error on the Scatter/Gather address, not 32byte aligned.

enumerator kEDMA_NbytesErrorFlag
NBYTES/CITER configuration error

enumerator kEDMA_DestinationOffsetErrorFlag
Destination offset not aligned with destination size

enumerator kEDMA_DestinationAddressErrorFlag
Destination address not aligned with destination size

enumerator kEDMA_SourceOffsetErrorFlag
Source offset not aligned with source size

enumerator kEDMA_SourceAddressErrorFlag
Source address not aligned with source size

enumerator kEDMA_ErrorChannelFlag
Error channel number of the cancelled channel number

enumerator kEDMA_TransferCanceledFlag
Transfer cancelled

enumerator kEDMA_ValidFlag
No error occurred, this bit is 0. Otherwise, it is 1.

_edma_interrupt_enable eDMA interrupt source
Values:

enumerator kEDMA_ErrorInterruptEnable
Enable interrupt while channel error occurs.

enumerator kEDMA_MajorInterruptEnable
Enable interrupt while major count exhausted.

enumerator kEDMA_HalfInterruptEnable
Enable interrupt while major count to half value.

enum _edma_transfer_type
eDMA transfer type
Values:

enumerator kEDMA_MemoryToMemory
Transfer from memory to memory

enumerator kEDMA_PeripheralToMemory
Transfer from peripheral to memory

enumerator kEDMA_MemoryToPeripheral
Transfer from memory to peripheral

enumerator kEDMA_PeripheralToPeripheral
Transfer from Peripheral to peripheral

enum edma_channel_memory_attribute
eDMA channel memory attribute
Values:

enumerator kEDMA_ChannelNoWriteNoReadNoCacheNoBuffer
No write allocate, no read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteNoReadNoCacheBufferable
No write allocate, no read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelNoWriteNoReadCacheableNoBuffer
No write allocate, no read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteNoReadCacheableBufferable
No write allocate, no read allocate, cacheable, bufferable.

enumerator kEDMA_ChannelNoWriteReadNoCacheNoBuffer
No write allocate, read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteReadNoCacheBufferable
No write allocate, read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelNoWriteReadCacheableNoBuffer
No write allocate, read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelNoWriteReadCacheableBufferable
No write allocate, read allocate, cacheable, bufferable.

enumerator kEDMA_ChannelWriteNoReadNoCacheNoBuffer
write allocate, no read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteNoReadNoCacheBufferable
write allocate, no read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelWriteNoReadCacheableNoBuffer
write allocate, no read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteNoReadCacheableBufferable
write allocate, no read allocate, cacheable, bufferable.

enumerator kEDMA_ChannelWriteReadNoCacheNoBuffer
write allocate, read allocate, non-cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteReadNoCacheBufferable
write allocate, read allocate, non-cacheable, bufferable.

enumerator kEDMA_ChannelWriteReadCacheableNoBuffer
write allocate, read allocate, cacheable, non-bufferable.

enumerator kEDMA_ChannelWriteReadCacheableBufferable
write allocate, read allocate, cacheable, bufferable.

enum _edma_channel_swap_size
eDMA4 channel swap size
Values:

enumerator kEDMA_ChannelSwapDisabled
Swap is disabled.

enumerator kEDMA_ChannelReadWith8bitSwap
Swap occurs with respect to the read 8bit.

enumerator kEDMA_ChannelReadWith16bitSwap
Swap occurs with respect to the read 16bit.

enumerator kEDMA_ChannelReadWith32bitSwap
Swap occurs with respect to the read 32bit.

enumerator kEDMA_ChannelWriteWith8bitSwap
Swap occurs with respect to the write 8bit.

enumerator kEDMA_ChannelWriteWith16bitSwap
Swap occurs with respect to the write 16bit.

enumerator kEDMA_ChannelWriteWith32bitSwap
Swap occurs with respect to the write 32bit.

eDMA channel system bus information, _edma_channel_sys_bus_info
Values:

enumerator kEDMA_PrivilegedAccessLevel
Privileged Access Level for DMA transfers. 0b - User protection level; 1b - Privileged protection level.

enumerator kEDMA_MasterId
DMA's master ID when channel is active and master ID replication is enabled.

enum _edma_channel_access_type
eDMA4 channel access type
Values:

enumerator kEDMA_ChannelDataAccess
Data access for eDMA4 transfers.

enumerator kEDMA_ChannelInstructionAccess
Instruction access for eDMA4 transfers.

enum _edma_channel_protection_level
eDMA4 channel protection level
Values:

enumerator kEDMA_ChannelProtectionLevelUser
user protection level for eDMA transfers.

enumerator kEDMA_ChannelProtectionLevelPrivileged
Privileged protection level eDMA transfers.

typedef enum *_edma_transfer_size* edma_transfer_size_t
eDMA transfer configuration

typedef enum *_edma_modulo* edma_modulo_t
eDMA modulo configuration

typedef enum *_edma_bandwidth* edma_bandwidth_t
Bandwidth control.

typedef enum *_edma_channel_link_type* edma_channel_link_type_t
Channel link type.

typedef enum *_edma_transfer_type* edma_transfer_type_t
eDMA transfer type

typedef struct *_edma_channel_Preemption_config* edma_channel_Preemption_config_t
eDMA channel priority configuration

typedef struct *_edma_minor_offset_config* edma_minor_offset_config_t
eDMA minor offset configuration

typedef enum *edma_channel_memory_attribute* edma_channel_memory_attribute_t
eDMA channel memory attribute

typedef enum *_edma_channel_swap_size* edma_channel_swap_size_t
eDMA4 channel swap size

typedef enum *_edma_channel_access_type* edma_channel_access_type_t
eDMA4 channel access type

typedef enum *_edma_channel_protection_level* edma_channel_protection_level_t
eDMA4 channel protection level

typedef struct *_edma_channel_config* edma_channel_config_t

eDMA4 channel configuration

typedef *edma_core_tcd_t* edma_tcd_t

eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

typedef struct *_edma_transfer_config* edma_transfer_config_t

eDMA channel transfer configuration

The transfer configuration structure support full feature configuration of the transfer control descriptor.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed enablchannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

---

Note: User should pay attention to the transfer size alignment limitation

a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0

b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width

c. the totalBytes should align with the bytesEachRequest

d. the srcAddr should align with the srcWidthOfEachTransfer

e. the dstAddr should align with the dstWidthOfEachTransfer

f. the srcAddr should align with srcAddrModulo if modulo feature is enabled

g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the eDMA interfaces will generate assert error.

---

typedef struct *_edma_config* edma_config_t

eDMA global configuration structure.

typedef void (*edma_callback)(struct *_edma_handle* *handle, void *userData, bool transferDone, uint32_t tcds)

Define callback function for eDMA.

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface EDMA_GetUnusedTCDNumber.

Param handle
EDMA handle pointer, users shall not touch the values inside.

Param userData
The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.

Param transferDone
If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDMA register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers.

Param tcds
How many tcds are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcds are finished between the last callback and this.

typedef struct *_edma_handle* edma_handle_t

eDMA transfer handle structure

FSL_EDMA_DRIVER_UNIFIED

eDMA driver name.

This is the unified eDMA driver serving the eDMA3, eDMA4 and eDMA5 IP versions. Its Kconfig component is driver.edma_unified; the historical driver.edma4 component and the FSL_EDMA_DRIVER_EDMA4 macro are retained as deprecated aliases for backward compatibility. The version-specific edma4_* / edma5_* identifiers inside this driver refer to the corresponding IP TCD layout, not to the component name.

FSL_EDMA_DRIVER_EDMA4

Deprecated alias of FSL_EDMA_DRIVER_UNIFIED; use FSL_EDMA_DRIVER_UNIFIED.

EDMA_ALLOCATE_TCD(name, number)

Macro used for allocate edma TCD.

DMA_DCHPRI_INDEX(channel)

Compute the offset unit from DCHPRI3.

struct _edma_channel_Preemption_config

#include <fsl_edma.h> eDMA channel priority configuration

Public Members

bool enableChannelPreemption

If true: a channel can be suspended by other channel with higher priority

bool enablePreemptAbility

If true: a channel can suspend other channel with low priority

uint8_t channelPriority
Channel priority

struct __edma__minor__offset__config
#include <fsl_edma.h> eDMA minor offset configuration

Public Members

bool enableSrcMinorOffset
Enable(true) or Disable(false) source minor loop offset.

bool enableDestMinorOffset
Enable(true) or Disable(false) destination minor loop offset.

uint32_t minorOffset
Offset for a minor loop mapping.

struct __edma__channel__config
#include <fsl_edma.h> eDMA4 channel configuration

Public Members

edma_channel_Preemption_config_t channelPreemptionConfig
channel preemption configuration

edma_channel_memory_attribute_t channelReadMemoryAttribute
channel memory read attribute configuration

edma_channel_memory_attribute_t channelWriteMemoryAttribute
channel memory write attribute configuration

edma_channel_swap_size_t channelSwapSize
channel swap size configuration

edma_channel_access_type_t channelAccessType
channel access type configuration

uint8_t channelDataSignExtensionBitPosition
channel data sign extension bit psition configuration

uint32_t channelRequestSource
hardware service request source for the channel

bool enableMasterIDReplication
enable master ID replication

edma_channel_protection_level_t protectionLevel

protection level

struct __edma__transfer__config

#include <fsl_edma.h> eDMA channel transfer configuration

The transfer configuration structure support full feature configuration of the transfer control descriptor.

1.To perform a simple transfer, below members should be initialized at least .srcAddr - source address .dstAddr - destination address .srcWidthOfEachTransfer - data width of source address .dstWidthOfEachTransfer - data width of destination address, normally it should be as same as srcWidthOfEachTransfer .bytesEachRequest - bytes to be transferred in each DMA request .totalBytes - total bytes to be transferred .srcOffsetOfEachTransfer - offset value in bytes unit to be applied to source address as each source read is completed .dstOffsetOfEachTransfer - offset value in bytes unit to be applied to destination address as each destination write is completed enablchannelRequest - channel request can be enabled together with transfer configure submission

2.The transfer configuration structure also support advance feature: Programmable source/destination address range(MODULO) Programmable minor loop offset Programmable major loop offset Programmable channel chain feature Programmable channel transfer control descriptor link feature

---

Note: User should pay attention to the transfer size alignment limitation

a. the bytesEachRequest should align with the srcWidthOfEachTransfer and the dstWidthOfEachTransfer that is to say bytesEachRequest % srcWidthOfEachTransfer should be 0

b. the srcOffsetOfEachTransfer and dstOffsetOfEachTransfer must be aligne with transfer width

c. the totalBytes should align with the bytesEachRequest

d. the srcAddr should align with the srcWidthOfEachTransfer

e. the dstAddr should align with the dstWidthOfEachTransfer

f. the srcAddr should align with srcAddrModulo if modulo feature is enabled

g. the dstAddr should align with dstAddrModulo if modulo feature is enabled If anyone of above condition can not be satisfied, the eDMA interfaces will generate assert error.

Public Members

uint32_t srcAddr

Source data address.

uint32_t destAddr

Destination data address.

edma_transfer_size_t srcTransferSize

Source data transfer size.

edma_transfer_size_t destTransferSize

Destination data transfer size.

int16_t srcOffset

Sign-extended offset value in byte unit applied to the current source address to form the next-state value as each source read is completed

int16_t destOffset

Sign-extended offset value in byte unit applied to the current destination address to form the next-state value as each destination write is completed.

uint32_t minorLoopBytes

bytes in each minor loop or each request range: 1 - (2^30 -1) when minor loop mapping is enabled range: 1 - (2^10 - 1) when minor loop mapping is enabled and source or dest minor loop offset is enabled range: 1 - (2^32 - 1) when minor loop mapping is disabled

uint32_t majorLoopCounts

minor loop counts in each major loop, should be 1 at least for each transfer range: (0 - (2^15 - 1)) when minor loop channel link is disabled range: (0 - (2^9 - 1)) when minor loop channel link is enabled total bytes in a transfer = minorLoopCountsEachMajor-Loop * bytesEachMinorLoop

uint16_t enabledInterruptMask

channel interrupt to enable, can be OR'ed value of _edma_interrupt_enable

edma_modulo_t srcAddrModulo

source circular data queue range

int32_t srcMajorLoopOffset

source major loop offset

edma_modulo_t dstAddrModulo

destination circular data queue range

int32_t dstMajorLoopOffset

destination major loop offset

bool enableSrcMinorLoopOffset

enable source minor loop offset

bool enableDstMinorLoopOffset

enable dest minor loop offset

int32_t minorLoopOffset

burst offset, the offset will be applied after minor loop update

bool enableChannelMajorLoopLink

channel link when major loop complete

uint32_t majorLoopLinkChannel

major loop link channel number

bool enableChannelMinorLoopLink

channel link when minor loop complete

uint32_t minorLoopLinkChannel

minor loop link channel number

edma_tcd_t *linkTCD

pointer to the link transfer control descriptor

struct _edma_config

#include <fsl_edma.h> eDMA global configuration structure.

Public Members

bool enableMasterIdReplication

Enable (true) master ID replication. If Master ID replication is disabled, the privileged protection level (supervisor mode) for eDMA4 transfers is used.

bool enableGlobalChannelLink

Enable(true) channel linking is available and controlled by each channel's link settings.

bool enableHaltOnError

Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

bool enableDebugMode

Enable(true) eDMA4 debug mode. When in debug mode, the eDMA4 stalls the start of a new channel. Executing channels are allowed to complete.

bool enableRoundRobinArbitration

Enable(true) channel linking is available and controlled by each channel's link settings.

edma_channel_config_t *channelConfig[1]

channel preemption configuration

struct _edma_handle

#include <fsl_edma.h> eDMA transfer handle structure

Public Members

edma_callback callback

Callback function for major count exhausted.

void *userData
Callback function parameter.

EDMA_ChannelType *channelBase
eDMA peripheral channel base address.

EDMA_Type *base
eDMA peripheral base address

EDMA_TCDType *tcdBase
eDMA peripheral tcd base address.

edma_tcd_t *tcdPool
Pointer to memory stored TCDs.

uint32_t channel
eDMA channel number.

volatile int8_t header
The first TCD index. Should point to the next TCD to be loaded into the eDMA engine.

volatile int8_t tail
The last TCD index. Should point to the next TCD to be stored into the memory pool.

volatile int8_t tcdUsed
The number of used TCD slots. Should reflect the number of TCDs can be used/loaded in the memory.

volatile int8_t tcdSize
The total number of TCD slots in the queue.

2.9 eDMA core Driver

enum _edma_tcd_type
eDMA tcd flag type
Values:

enumerator kEDMA_EDMA4Flag
Data access for eDMA4 transfers.

enumerator kEDMA_EDMA5Flag
Instruction access for eDMA4 transfers.

typedef struct *_edma_core_mp* edma_core_mp_t
edma core channel struture definition

typedef struct *_edma_core_channel* edma_core_channel_t
edma core channel struture definition

typedef enum *_edma_tcd_type* edma_tcd_type_t
eDMA tcd flag type

typedef struct *_edma5_core_tcd* edma5_core_tcd_t
edma5 core TCD struture definition

typedef struct *_edma4_core_tcd* edma4_core_tcd_t
edma4 core TCD struture definition

typedef struct *_edma_core_tcd* edma_core_tcd_t
edma core TCD struture definition

typedef *edma_core_channel_t* EDMA_ChannelType
EDMA typedef.

typedef *edma_core_tcd_t* EDMA_TCDType

typedef void EDMA_Type

DMA_CORE_MP_CSR_EDBG_MASK

DMA_CORE_MP_CSR_ERCA_MASK

DMA_CORE_MP_CSR_HAE_MASK

DMA_CORE_MP_CSR_HALT_MASK

DMA_CORE_MP_CSR_GCLC_MASK

DMA_CORE_MP_CSR_GMRC_MASK

DMA_CORE_MP_CSR_EDBG(**x**)

DMA_CORE_MP_CSR_ERCA(**x**)

DMA_CORE_MP_CSR_HAE(**x**)

DMA_CORE_MP_CSR_HALT(**x**)

DMA_CORE_MP_CSR_GCLC(**x**)

DMA_CORE_MP_CSR_GMRC(**x**)

DMA_CSR_INTMAJOR_MASK

DMA_CSR_INTHALF_MASK

DMA_CSR_DREQ_MASK

DMA_CSR_ESG_MASK

DMA_CSR_BWC_MASK

DMA_CSR_BWC(x)

DMA_CSR_START_MASK

DMA_CITER_ELINKNO_CITER_MASK

DMA_BITER_ELINKNO_BITER_MASK

DMA_CITER_ELINKNO_CITER_SHIFT

DMA_CITER_ELINKYES_CITER_MASK

DMA_CITER_ELINKYES_CITER_SHIFT

DMA_ATTR_SMOD_MASK

DMA_ATTR_DMOD_MASK

DMA_ATTR_SSIZE_MASK

DMA_ATTR_SSIZE_SHIFT

DMA_ATTR_DSIZE_MASK

DMA_ATTR_DSIZE_SHIFT

DMA_CITER_ELINKNO_ELINK_MASK

DMA_CSR_MAJORELINK_MASK

DMA_BITER_ELINKYES_ELINK_MASK

DMA_CITER_ELINKYES_ELINK_MASK

DMA_CSR_MAJORLINKCH_MASK

DMA_BITER_ELINKYES_LINKCH_MASK

DMA_CITER_ELINKYES_LINKCH_MASK

DMA__NBYTES__MLOFFYES__MLOFF__MASK

DMA__NBYTES__MLOFFYES__DMLOE__MASK

DMA__NBYTES__MLOFFYES__SMLOE__MASK

DMA__NBYTES__MLOFFNO__NBYTES__MASK

DMA__ATTR__DMOD(**x**)

DMA__ATTR__SMOD(**x**)

DMA__BITER__ELINKYES__LINKCH(**x**)

DMA__CITER__ELINKYES__LINKCH(**x**)

DMA__NBYTES__MLOFFYES__MLOFF(**x**)

DMA__NBYTES__MLOFFYES__DMLOE(**x**)

DMA__NBYTES__MLOFFYES__SMLOE(**x**)

DMA__NBYTES__MLOFFNO__NBYTES(**x**)

DMA__NBYTES__MLOFFYES__NBYTES(**x**)

DMA__ATTR__DSIZE(**x**)

DMA__ATTR__SSIZE(**x**)

DMA__CSR__DREQ(**x**)

DMA__CSR__MAJORLINKCH(**x**)

DMA__CH__MATTR__WCACHE(**x**)

DMA__CH__MATTR__RCACHE(**x**)

DMA__CH__CSR__SIGNEXT__MASK

DMA__CH__CSR__SIGNEXT__SHIFT

DMA__CH__CSR__SWAP__MASK

DMA__CH__CSR__SWAP__SHIFT

DMA__CH__SBR__INSTR__MASK

DMA__CH__SBR__INSTR__SHIFT

DMA__CH__SBR__EMI__MASK

DMA__CH__SBR__EMI__SHIFT

DMA__CH__MUX__SOURCE(**x**)

DMA_ERR_DBE_FLAG
DMA error flag.

DMA_ERR_SBE_FLAG

DMA_ERR_SGE_FLAG

DMA_ERR_NCE_FLAG

DMA_ERR_DOE_FLAG

DMA_ERR_DAE_FLAG

DMA_ERR_SOE_FLAG

DMA_ERR_SAE_FLAG

DMA_ERR_ERRCHAN_FLAG

DMA_ERR_ECX_FLAG

DMA_ERR_FLAG

DMA_CLEAR_DONE_STATUS(base, channel)
get/clear DONE bit

DMA_GET_DONE_STATUS(base, channel)

DMA_ENABLE_ERROR_INT(base, channel)
enable/disable error interupt

DMA_DISABLE_ERROR_INT(base, channel)

DMA_CLEAR_ERROR_STATUS(base, channel)
get/clear error status

DMA_GET_ERROR_STATUS(base, channel)

DMA_CLEAR_INT_STATUS(base, channel)
get/clear INT status

DMA_GET_INT_STATUS(base, channel)

DMA_ENABLE_MAJOR_INT(base, channel)
enable/dsiable MAJOR/HALF INT

DMA_ENABLE_HALF_INT(base, channel)

DMA_DISABLE_MAJOR_INT(base, channel)

DMA_DISABLE_HALF_INT(base, channel)

EDMA_TCD_ALIGN_SIZE
EDMA tcd align size.

EDMA_CORE_BASE(base)
EDMA base address convert macro.

EDMA_MP_BASE(base)

EDMA_CHANNEL_BASE(base, channel)

EDMA_TCD_BASE(base, channel)

EDMA_TCD_TYPE(x)
EDMA TCD type macro.

EDMA_TCD_SADDR(tcd, flag)
EDMA TCD address convert macro.

EDMA_TCD_SOFF(tcd, flag)

EDMA_TCD_ATTR(tcd, flag)

EDMA_TCD_NBYTES(tcd, flag)

EDMA_TCD_SLAST(tcd, flag)

EDMA_TCD_DADDR(tcd, flag)

EDMA_TCD_DOFF(tcd, flag)

EDMA_TCD_CITER(tcd, flag)

EDMA_TCD_DLAST_SGA(tcd, flag)

EDMA_TCD_CSR(tcd, flag)

EDMA_TCD_BITER(tcd, flag)

struct _edma_core_mp
#include <fsl_edma_core.h> edma core channel struture definition

Public Members

___IO uint32_t MP_CSR
Channel Control and Status, array offset: 0x10000, array step: 0x10000

___IO uint32_t MP_ES
Channel Error Status, array offset: 0x10004, array step: 0x10000

struct _edma_core_channel
#include <fsl_edma_core.h> edma core channel struture definition

Public Members

___IO uint32_t CH_CSR
Channel Control and Status, array offset: 0x10000, array step: 0x10000

___IO uint32_t CH_ES
Channel Error Status, array offset: 0x10004, array step: 0x10000

___IO uint32_t CH_INT
Channel Interrupt Status, array offset: 0x10008, array step: 0x10000

___IO uint32_t CH_SBR
Channel System Bus, array offset: 0x1000C, array step: 0x10000

___IO uint32_t CH_PRI
Channel Priority, array offset: 0x10010, array step: 0x10000

struct _edma5_core_tcd
#include <fsl_edma_core.h> edma5 core TCD struture definition

Public Members

___IO uint32_t SADDR
SADDR register, used to save source address

___IO uint32_t SADDR_HIGH
SADDR HIGH register, used to save source address

___IO uint16_t SOFF
SOFF register, save offset bytes every transfer

___IO uint16_t ATTR
ATTR register, source/destination transfer size and modulo

___IO uint32_t NBYTES
Nbytes register, minor loop length in bytes

___IO uint32_t SLAST
SLAST register

___IO uint32_t SLAST_SDA_HIGH
SLAST SDA HIGH register

___IO uint32_t DADDR
DADDR register, used for destination address

___IO uint32_t DADDR_HIGH
DADDR HIGH register, used for destination address

___IO uint32_t DLAST_SGA
DLASTSGA register, next tcd address used in scatter-gather mode

___IO uint32_t DLAST_SGA_HIGH
DLASTSGA HIGH register, next tcd address used in scatter-gather mode

___IO uint16_t DOFF
DOFF register, used for destination offset

___IO uint16_t CITER
CITER register, current minor loop numbers, for unfinished minor loop.

___IO uint16_t CSR
CSR register, for TCD control status

___IO uint16_t BITER
BITER register, begin minor loop count.

uint8_t RESERVED[16]
Aligned 64 bytes

struct _edma4_core_tcd
#include <fsl_edma_core.h> edma4 core TCD struture definition

Public Members

___IO uint32_t SADDR
SADDR register, used to save source address

___IO uint16_t SOFF
SOFF register, save offset bytes every transfer

___IO uint16_t ATTR
ATTR register, source/destination transfer size and modulo

___IO uint32_t NBYTES
Nbytes register, minor loop length in bytes

___IO uint32_t SLAST
SLAST register

___IO uint32_t DADDR
DADDR register, used for destination address

___IO uint16_t DOFF
DOFF register, used for destination offset

___IO uint16_t CITER
CITER register, current minor loop numbers, for unfinished minor loop.

___IO uint32_t DLAST_SGA
DLASTSGA register, next tcd address used in scatter-gather mode

___IO uint16_t CSR
CSR register, for TCD control status

___IO uint16_t BITER
BITER register, begin minor loop count.

struct _edma_core_tcd
#include <fsl_edma_core.h> edma core TCD struture definition

union MP_REGS

Public Members

struct *_edma_core_mp* EDMA5_REG

struct EDMA5_REG

Public Members

___IO uint32_t MP_INT_LOW
Channel Control and Status, array offset: 0x10008, array step: 0x10000

___I uint32_t MP_INT_HIGH
Channel Control and Status, array offset: 0x1000C, array step: 0x10000

___I uint32_t MP_HRS_LOW
Channel Control and Status, array offset: 0x10010, array step: 0x10000

___I uint32_t MP_HRS_HIGH
Channel Control and Status, array offset: 0x10014, array step: 0x10000

___IO uint32_t MP_STOPCH
Channel Control and Status, array offset: 0x10020, array step: 0x10000

___I uint32_t MP_SSR_LOW
Channel Control and Status, array offset: 0x10030, array step: 0x10000

___I uint32_t MP_SSR_HIGH
Channel Control and Status, array offset: 0x10034, array step: 0x10000

___IO uint32_t CH_GRPRI [64]
Channel Control and Status, array offset: 0x10100, array step: 0x10000

___IO uint32_t CH_MUX [64]
Channel Control and Status, array offset: 0x10200, array step: 0x10000

___IO uint32_t CH_PROT [64]
Channel Control and Status, array offset: 0x10400, array step: 0x10000

union CH_REGS

Public Members

struct *_edma_core_channel* EDMA5_REG

struct *_edma_core_channel* EDMA4_REG

struct EDMA5_REG

Public Members

___IO uint32_t CH_MATTR
Memory Attributes Register, array offset: 0x10018, array step: 0x8000

struct EDMA4_REG

Public Members

___IO uint32_t CH_MUX
Channel Multiplexor Configuration, array offset: 0x10014, array step: 0x10000

___IO uint16_t CH_MATTR
Memory Attributes Register, array offset: 0x10018, array step: 0x8000

union TCD_REGS

Public Members

edma4_core_tcd_t edma4_tcd

2.10 eDMA soc Driver

void EDMA_EnableRequest(DMA_Type *base, dma_request_source_t requestSource)
Enable the eDMA request.

This function enables the eDMA request controlled by SYSCON.

Parameters

- base – eDMA peripheral base address.
- requestSource – eDMA request source, see “dma_request_source_t”.

void EDMA_DisableRequest(DMA_Type *base, dma_request_source_t requestSource)

Disable the eDMA request.

This function disables the eDMA request controlled by SYSCON.

Parameters

- base – eDMA peripheral base address.
- requestSource – eDMA request source, see “dma_request_source_t”.

FSL_EDMA_SOC_DRIVER_VERSION

Driver version 1.0.0.

FSL_EDMA_SOC_IP_DMA3

DMA IP version.

FSL_EDMA_SOC_IP_DMA4

EDMA_CHANNEL_OFFSET

EDMA base address convert macro.

EDMA_CHANNEL_ARRAY_STEP(base)

2.11 Ezhv

enum _arm2ezhv_intctl

Definition of 4 interrupt requests from ARM to EZH-V.

Values:

enumerator kEZHV_ARM2EZHV_MEI

enumerator kEZHV_ARM2EZHV_SEI

enumerator kEZHV_ARM2EZHV_MSI

enumerator kEZHV_ARM2EZHV_MTI

enum _ezhv2arm_int_chan

EZH-V to ARM interrupt channel.

Values:

enumerator kEZHV_EzhvToArmIntChan0

enumerator kEZHV_EzhvToArmIntChan1

enumerator kEZHV_EzhvToArmIntChan2

enumerator kEZHV_EzhvToArmIntChan3

enumerator kEZHV_EzhvToArmIntChan4

enumerator kEZHV_EzhvToArmIntChan5

enumerator kEZHV_EzhvToArmIntChan6

enumerator kEZHV_EzhvToArmIntChan7

enumerator kEZHV_EzhvToArmIntChan8

enumerator kEZHV_EzhvToArmIntChan9

enumerator kEZHV_EzhvToArmIntChan10

enumerator kEZHV_EzhvToArmIntChan11

enumerator kEZHV_EzhvToArmIntChan12

enumerator kEZHV_EzhvToArmIntChan13

enumerator kEZHV_EzhvToArmIntChan14

enumerator kEZHV_EzhvToArmIntChan15

typedef void (*ezhv_callback_t)(void *userData)

define callback function for EZH-V

This callback function is called in EZH-V interrupt handle.

typedef struct *_ezhv_copy_image* ezhv_copy_image_t

Structure for EZH-V copy image to destination address.

Defines source and destination address for copying image with given size.

typedef uint32_t ezhv_api_t

@Brief List of EZH-V APIs

typedef struct *_ezhv_param* ezhv_param_t

parameters used by EZH-V

typedef enum *_arm2ezhv_intctl* arm2ezhv_intctl_t

Definition of 4 interrupt requests from ARM to EZH-V.

typedef enum *_ezhv2arm_int_chan* ezhv2arm_int_chan_t

EZH-V to ARM interrupt channel.

void EZHV_Init(*ezhv_copy_image_t* *ezhvCopyImage)

Initialize the EZH-V.

Parameters

- ezhvCopyImage – The information about the EZH-V image to copy.

void EZHV_InitWithoutFirmware(void)

Initialize the EZH-V.

This function is similar with EZHV_Init, the difference is this function does not install the firmware, the firmware could be installed using EZHV_InstallFirmware.

void EZHV_InstallFirmware(*ezhv_copy_image_t* *ezhvCopyImage)

install EZH-V firmware by given image info

Parameters

- ezhvCopyImage – The information about the EZH-V image to copy.

void EZHV_Boot(uint32_t bootAddr)

Boot EZH-V from given address bootAddr.

Parameters

- bootAddr – The boot address.

void EZHV_Deinit(void)

Deinitialize the EZH-V.

void EZHV_SetCallback(*ezhv_callback_t* callback, uint16_t channel, void *userData)

Install the EZH-V callback function.

Parameters

- callback – EZH-V callback function pointer.
- channel – interrupt channel index.
- userData – A parameter for the callback function.

void EZHV_SetPara(*ezhv_param_t* *para)

Set the parameter used by EZH-V.

Parameters

- para – Parameter written into shared mem between ARM and EZH-V.

uint32_t *EZHV_GetParaAddr(void)

Get shared space address.

Returns

The start address of shared space

void EZHV_WakeUpEzhv(*arm2ezhv_intctl_t* arm2ezhvInt)

Check the wait status and wake up EZH-V.

Parameters

- arm2ezhvInt – The interrupt request which will send to the EZH-V.

static inline void EZHV_EnableArm2EzhvInt(*arm2ezhv_intctl_t* arm2ezhvInt)

Enable interrupt request from ARM core to EZH-V core.

Note: This interrupt can be used to send interrupt request to EZH-V

Parameters

- arm2ezhvInt – The interrupt request which will send to the EZH-V.

static inline void EZHV_DisableArm2EzhvInt(*arm2ezhv_intctl_t* arm2ezhvInt)

Disable interrupt request from ARM core to EZH-V core.

Parameters

- arm2ezhvInt – The interrupt request type.

void EZHV_DriverIRQHandler(void)

EZH-V IRQ handler.

static inline void EZHV_EnableEzhv2ArmIntChan(*ezhv2arm_int_chan_t* chan)

Enable the EZH-V interrupt channel to ARM core.

Parameters

- chan – Ezhv to arm interrupt channel index.

static inline void EZHV_DisableEzhv2ArmIntChan(*ezhv2arm_int_chan_t* chan)

Disable the EZH-V interrupt channel to ARM core.

Parameters

- chan – Ezhv to arm interrupt channel index.

static inline uint32_t EZHV_GetEzhv2ArmIntChan(void)

Get EZH-V interrupt outputs.

static inline void EZHV_ClearEzhv2ArmIntChan(*ezhv2arm_int_chan_t* chan)

Clear EZH-V interrupt outputs.

Parameters

- chan – EZH-V to ARM interrupt channel index.

static inline bool EZHV_GetEzhvStopStatusFlag(void)

Get EZH-V stop status flag.

static inline bool EZHV_GetEzhvHaltStatusFlag(void)

Get EZH-V halt status flag.

static inline bool EZHV_GetEzhvWaitStatusFlag(void)

Get EZH-V wait status flag.

static inline bool EZHV_GetEzhvWakeupStatusFlag(void)

Get EZH-V wakeup status flag.

FSL_EZHV_DRIVER_VERSION

cache driver version.

EZHV_API_MAX_ARG

uint32_t srcAddr

uint32_t destAddr

uint32_t size

ezhv_api_t ezhvApi

uint32_t argc

uint32_t *argv[8U]

EZHV_M_EXT_INT
EZHV inside machine-mode external isr offset.

EZHV_SHARED_DATA_ADDR
shared data space between ARM and EZHV, space size is 256 Bytes

EZHV_INT_CHAN_NUM
definition for active ezhv2arm int channel number

please redefine this macro's value if less channels are enabled

struct _ezhv_copy_image
#include <fsl_ezhv.h> Structure for EZH-V copy image to destination address.

Defines source and destination address for copying image with given size.

struct _ezhv_param
#include <fsl_ezhv.h> parameters used by EZH-V

2.12 FGPIO Driver

2.13 FlexIO: FlexIO Driver

2.14 FlexIO Camera Driver

void FLEXIO_CAMERA_Init(*FLEXIO_CAMERA_Type* *base, const *flexio_camera_config_t* *config)
Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO Camera.

Parameters

- base – Pointer to FLEXIO_CAMERA_Type structure
- config – Pointer to flexio_camera_config_t structure

void FLEXIO_CAMERA_Deinit(*FLEXIO_CAMERA_Type* *base)
Resets the FLEXIO_CAMERA shifer and timer config.

Note: After calling this API, call FLEXO_CAMERA_Init to use the FlexIO Camera module.

Parameters

- base – Pointer to FLEXIO_CAMERA_Type structure

void FLEXIO_CAMERA_GetDefaultConfig(*flexio_camera_config_t* *config)

Gets the default configuration to configure the FlexIO Camera. The configuration can be used directly for calling the *FLEXIO_CAMERA_Init()*. Example:

```
flexio_camera_config_t config;
FLEXIO_CAMERA_GetDefaultConfig(&userConfig);
```

Parameters

- config – Pointer to the flexio_camera_config_t structure

static inline void FLEXIO_CAMERA_Enable(*FLEXIO_CAMERA_Type* *base, bool enable)

Enables/disables the FlexIO Camera module operation.

Parameters

- base – Pointer to the FLEXIO_CAMERA_Type
- enable – True to enable, false does not have any effect.

uint32_t FLEXIO_CAMERA_GetStatusFlags(*FLEXIO_CAMERA_Type* *base)

Gets the FlexIO Camera status flags.

Parameters

- base – Pointer to FLEXIO_CAMERA_Type structure

Returns

FlexIO shifter status flags

- FLEXIO_SHIFTSTAT_SSF_MASK
- 0

void FLEXIO_CAMERA_ClearStatusFlags(*FLEXIO_CAMERA_Type* *base, uint32_t mask)

Clears the receive buffer full flag manually.

Parameters

- base – Pointer to the device.
- mask – status flag The parameter can be any combination of the following values:
  - kFLEXIO_CAMERA_RxDataRegFullFlag
  - kFLEXIO_CAMERA_RxErrorFlag

void FLEXIO_CAMERA_EnableInterrupt(*FLEXIO_CAMERA_Type* *base)

Switches on the interrupt for receive buffer full event.

Parameters

- base – Pointer to the device.

void FLEXIO_CAMERA_DisableInterrupt(*FLEXIO_CAMERA_Type* *base)

Switches off the interrupt for receive buffer full event.

Parameters

- base – Pointer to the device.

static inline void FLEXIO_CAMERA_EnableRxDMA(*FLEXIO_CAMERA_Type* *base, bool enable)

Enables/disables the FlexIO Camera receive DMA.

The FlexIO Camera mode can't work without the DMA or eDMA support, Usually, it needs at least two DMA or eDMA channels, one for transferring data from Camera, such as 0V7670 to FlexIO buffer, another is for transferring data from FlexIO buffer to LCD.

Parameters

- base – Pointer to FLEXIO_CAMERA_Type structure
- enable – True to enable, false to disable.

static inline uint32_t FLEXIO_CAMERA_GetRxBufferAddress(*FLEXIO_CAMERA_Type* *base)

Gets the data from the receive buffer.

Parameters

- base – Pointer to the device.

Returns

data Pointer to the buffer that keeps the data with count of base->shifterCount .

FSL_FLEXIO_CAMERA_DRIVER_VERSION

FlexIO Camera driver version.

Error codes for the Camera driver.

Values:

enumerator kStatus_FLEXIO_CAMERA_RxBusy

Receiver is busy.

enumerator kStatus_FLEXIO_CAMERA_RxIdle

Camera receiver is idle.

enum _flexio_camera_status_flags

Define FlexIO Camera status mask.

Values:

enumerator kFLEXIO_CAMERA_RxDataRegFullFlag

Receive buffer full flag.

enumerator kFLEXIO_CAMERA_RxErrorFlag

Receive buffer error flag.

typedef struct *_flexio_camera_type* FLEXIO_CAMERA_Type

Define structure of configuring the FlexIO Camera device.

typedef struct *_flexio_camera_config* flexio_camera_config_t

Define FlexIO Camera user configuration structure.

typedef struct *_flexio_camera_transfer* flexio_camera_transfer_t

Define FlexIO Camera transfer structure.

FLEXIO_CAMERA_PARALLEL_DATA_WIDTH

Define the Camera CPI interface is constantly 8-bit width.

struct _flexio_camera_type

#include <fsl_flexio_camera.h> Define structure of configuring the FlexIO Camera device.

Public Members

FLEXIO_Type *flexioBase

FlexIO module base address.

uint32_t datPinStartIdx

First data pin (D0) index for flexio_camera. Then the successive following FLEXIO_CAMERA_DATA_WIDTH-1 pins are used as D1-D7.

uint32_t pclkPinIdx

Pixel clock pin (PCLK) index for flexio_camera.

uint32_t hrefPinIdx

Horizontal sync pin (HREF) index for flexio_camera.

uint8_t shifterStartIdx

First shifter index used for flexio_camera data FIFO.

uint32_t shifterCount

The count of shifters that are used as flexio_camera data FIFO.

uint32_t timerIdx

Timer index used for flexio_camera in FlexIO.

struct _flexio_camera_config

#include <fsl_flexio_camera.h> Define FlexIO Camera user configuration structure.

Public Members

bool enablecamera

Enable/disable FlexIO Camera TX & RX.

bool enableInDoze

Enable/disable FlexIO operation in doze mode

bool enableInDebug

Enable/disable FlexIO operation in debug mode

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

struct _flexio_camera_transfer

#include <fsl_flexio_camera.h> Define FlexIO Camera transfer structure.

Public Members

uint32_t dataAddress

Transfer buffer

uint32_t dataNum

Transfer num

2.15 FlexIO Driver

void FLEXIO_GetDefaultConfig(*flexio_config_t* *userConfig)

Gets the default configuration to configure the FlexIO module. The configuration can used directly to call the FLEXIO_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

Parameters

- userConfig – pointer to flexio_config_t structure

void FLEXIO_Init(FLEXIO_Type *base, const *flexio_config_t* *userConfig)

Configures the FlexIO with a FlexIO configuration. The configuration structure can be filled by the user or be set with default values by *FLEXIO_GetDefaultConfig()*.

Example

```
flexio_config_t config = {
.enableFlexio = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false
};
FLEXIO_Configure(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- userConfig – pointer to flexio_config_t structure

void FLEXIO_Deinit(FLEXIO_Type *base)

Gates the FlexIO clock. Call this API to stop the FlexIO clock.

Note: After calling this API, call the FLEXO_Init to use the FlexIO module.

Parameters

- base – FlexIO peripheral base address

uint32_t FLEXIO_GetInstance(FLEXIO_Type *base)

Get instance number for FLEXIO module.

Parameters

- base – FLEXIO peripheral base address.

void FLEXIO_Reset(FLEXIO_Type *base)

Resets the FlexIO module.

Parameters

- base – FlexIO peripheral base address

static inline void FLEXIO_Enable(FLEXIO_Type *base, bool enable)

Enables the FlexIO module operation.

Parameters

- base – FlexIO peripheral base address
- enable – true to enable, false to disable.

static inline uint32_t FLEXIO_ReadPinInput(FLEXIO_Type *base)

Reads the input data on each of the FlexIO pins.

Parameters

- base – FlexIO peripheral base address

Returns

FlexIO pin input data

static inline uint8_t FLEXIO_GetShifterState(FLEXIO_Type *base)

Gets the current state pointer for state mode use.

Parameters

- base – FlexIO peripheral base address

Returns

current State pointer

void FLEXIO_SetShifterConfig(FLEXIO_Type *base, uint8_t index, const *flexio_shifter_config_t* *shifterConfig)

Configures the shifter with the shifter configuration. The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

Example

```
flexio_shifter_config_t config = {
.timerSelect = 0,
.timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,
```

(continues on next page)

(continued from previous page)

```
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinPolarity = kFLEXIO_PinActiveLow,
.shifterMode = kFLEXIO_ShifterModeTransmit,
.inputSource = kFLEXIO_ShifterInputFromPin,
.shifterStop = kFLEXIO_ShifterStopBitHigh,
.shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Shifter index
- shifterConfig – Pointer to flexio_shifter_config_t structure

void FLEXIO_SetTimerConfig(FLEXIO_Type *base, uint8_t index, const *flexio_timer_config_t* *timerConfig)

Configures the timer with the timer configuration. The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

Example

```
flexio_timer_config_t config = {
.triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(0),
.triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
.triggerSource = kFLEXIO_TimerTriggerSourceInternal,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinSelect = 0,
.pinPolarity = kFLEXIO_PinActiveHigh,
.timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
.timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
.timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput,
.timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
.timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
.timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
.timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
.timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

Parameters

- base – FlexIO peripheral base address
- index – Timer index
- timerConfig – Pointer to the flexio_timer_config_t structure

static inline void FLEXIO_SetClockMode(FLEXIO_Type *base, uint8_t index, *flexio_timer_decrement_source_t* clocksource)

This function set the value of the prescaler on flexio channels.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- index – Timer index
- clocksource – Set clock value

static inline void FLEXIO_EnableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the shifter status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_DisableShifterStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the shifter status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_EnableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the shifter error interrupt. The interrupt generates when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_DisableShifterErrorInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the shifter error interrupt. The interrupt won't generate when the corresponding SEF is set.

Note: For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by (1 « shifter index)

static inline void FLEXIO_EnableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Enables the timer status interrupt. The interrupt generates when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using ((1 « timer index0) | (1 « timer index1))

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by (1 « timer index)

static inline void FLEXIO_DisableTimerStatusInterrupts(FLEXIO_Type *base, uint32_t mask)

Disables the timer status interrupt. The interrupt won't generate when the corresponding SSF is set.

Note: For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using ((1 « timer index0) | (1 « timer index1))

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by (1 « timer index)

static inline uint32_t FLEXIO_GetShifterStatusFlags(FLEXIO_Type *base)

Gets the shifter status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter status flags

static inline void FLEXIO_ClearShifterStatusFlags(FLEXIO_Type *base, uint32_t mask)

Clears the shifter status flags.

Note: For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)

static inline uint32_t FLEXIO_GetShifterErrorFlags(FLEXIO_Type *base)

Gets the shifter error flags.

Parameters

- base – FlexIO peripheral base address

Returns

Shifter error flags

static inline void FLEXIO_ClearShifterErrorFlags(FLEXIO_Type *base, uint32_t mask)

Clears the shifter error flags.

Note: For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter error mask which can be calculated by (1 « shifter index)

static inline uint32_t FLEXIO_GetTimerStatusFlags(FLEXIO_Type *base)

Gets the timer status flags.

Parameters

- base – FlexIO peripheral base address

Returns

Timer status flags

static inline void FLEXIO_ClearTimerStatusFlags(FLEXIO_Type *base, uint32_t mask)

Clears the timer status flags.

Note: For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using ((1 « timer index0) | (1 « timer index1))

Parameters

- base – FlexIO peripheral base address
- mask – The timer status mask which can be calculated by (1 « timer index)

static inline void FLEXIO_EnableShifterStatusDMA(FLEXIO_Type *base, uint32_t mask, bool enable)

Enables/disables the shifter status DMA. The DMA request generates when the corresponding SSF is set.

Note: For multiple shifter status DMA enables, for example, calculate the mask by using ((1 « shifter index0) | (1 « shifter index1))

Parameters

- base – FlexIO peripheral base address
- mask – The shifter status mask which can be calculated by (1 « shifter index)
- enable – True to enable, false to disable.

uint32_t FLEXIO_GetShifterBufferAddress(FLEXIO_Type *base, *flexio_shifter_buffer_type_t* type, uint8_t index)

Gets the shifter buffer address for the DMA transfer usage.

Parameters

- base – FlexIO peripheral base address

- type – Shifter type of flexio_shifter_buffer_type_t
- index – Shifter index

Returns
Corresponding shifter buffer index

status_t FLEXIO_RegisterHandleIRQ(void *base, void *handle, *flexio_isr_t* isr)
Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.
- handle – Pointer to the handler for FlexIO simulated peripheral.
- isr – FlexIO simulated peripheral interrupt handler.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_UnregisterHandleIRQ(void *base)
Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.

Parameters

- base – Pointer to the FlexIO simulated peripheral type.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

static inline void FLEXIO_ClearPortOutput(FLEXIO_Type *base, uint32_t mask)
Sets the output level of the multiple FLEXIO pins to the logic 0.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_SetPortOutput(FLEXIO_Type *base, uint32_t mask)
Sets the output level of the multiple FLEXIO pins to the logic 1.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_TogglePortOutput(FLEXIO_Type *base, uint32_t mask)
Reverses the current output logic of the multiple FLEXIO pins.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

static inline void FLEXIO_PinWrite(FLEXIO_Type *base, uint32_t pin, uint8_t output)
Sets the output level of the FLEXIO pins to the logic 1 or 0.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

- output – FLEXIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void FLEXIO_EnablePinOutput(FLEXIO_Type *base, uint32_t pin)

Enables the FLEXIO output pin function.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

static inline uint32_t FLEXIO_PinRead(FLEXIO_Type *base, uint32_t pin)

Reads the current input value of the FLEXIO pin.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

static inline uint32_t FLEXIO_GetPinStatus(FLEXIO_Type *base, uint32_t pin)

Gets the FLEXIO input pin status.

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.

Return values

FLEXIO – port input status

- 0: corresponding pin input capture no status.
- 1: corresponding pin input capture rising or falling edge.

static inline void FLEXIO_SetPinLevel(FLEXIO_Type *base, uint8_t pin, bool level)

Sets the FLEXIO output pin level.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.
- level – FlexIO output pin level to set, can be either 0 or 1.

static inline bool FLEXIO_GetPinOverride(const FLEXIO_Type *const base, uint8_t pin)

Gets the enabled status of a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – FlexIO pin number.

Return values

FlexIO – port enabled status

- 0: corresponding output pin is in disabled state.
- 1: corresponding output pin is in enabled state.

static inline void FLEXIO_ConfigPinOverride(FLEXIO_Type *base, uint8_t pin, bool enabled)

Enables or disables a FLEXIO output pin.

Parameters

- base – FlexIO peripheral base address
- pin – Flexio pin number.
- enabled – Enable or disable the FlexIO pin.

static inline void FLEXIO_ClearPortStatus(FLEXIO_Type *base, uint32_t mask)

Clears the multiple FLEXIO input pins status.

Parameters

- base – FlexIO peripheral base address
- mask – FLEXIO pin number mask

FSL_FLEXIO_DRIVER_VERSION

FlexIO driver version.

enum _flexio_timer_trigger_polarity

Define time of timer trigger polarity.

Values:

enumerator kFLEXIO_TimerTriggerPolarityActiveHigh

Active high.

enumerator kFLEXIO_TimerTriggerPolarityActiveLow

Active low.

enum _flexio_timer_trigger_source

Define type of timer trigger source.

Values:

enumerator kFLEXIO_TimerTriggerSourceExternal

External trigger selected.

enumerator kFLEXIO_TimerTriggerSourceInternal

Internal trigger selected.

enum _flexio_pin_config

Define type of timer/shifter pin configuration.

Values:

enumerator kFLEXIO_PinConfigOutputDisabled

Pin output disabled.

enumerator kFLEXIO_PinConfigOpenDrainOrBidirection

Pin open drain or bidirectional output enable.

enumerator kFLEXIO_PinConfigBidirectionOutputData
Pin bidirectional output data.

enumerator kFLEXIO_PinConfigOutput
Pin output.

enum _flexio_pin_polarity
Definition of pin polarity.
Values:

enumerator kFLEXIO_PinActiveHigh
Active high.

enumerator kFLEXIO_PinActiveLow
Active low.

enum _flexio_timer_mode
Define type of timer work mode.
Values:

enumerator kFLEXIO_TimerModeDisabled
Timer Disabled.

enumerator kFLEXIO_TimerModeDual8BitBaudBit
Dual 8-bit counters baud/bit mode.

enumerator kFLEXIO_TimerModeDual8BitPWM
Dual 8-bit counters PWM mode.

enumerator kFLEXIO_TimerModeSingle16Bit
Single 16-bit counter mode.

enumerator kFLEXIO_TimerModeDual8BitPWMLow
Dual 8-bit counters PWM Low mode.

enum _flexio_timer_output
Define type of timer initial output or timer reset condition.
Values:

enumerator kFLEXIO_TimerOutputOneNotAffectedByReset
Logic one when enabled and is not affected by timer reset.

enumerator kFLEXIO_TimerOutputZeroNotAffectedByReset
Logic zero when enabled and is not affected by timer reset.

enumerator kFLEXIO_TimerOutputOneAffectedByReset
Logic one when enabled and on timer reset.

enumerator kFLEXIO_TimerOutputZeroAffectedByReset
Logic zero when enabled and on timer reset.

enum _flexio_timer_decrement_source
Define type of timer decrement.
Values:

enumerator kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput
Decrement counter on FlexIO clock, Shift clock equals Timer output.

enumerator kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput
Decrement counter on Trigger input (both edges), Shift clock equals Timer output.

enumerator kFLEXIO_TimerDecSrcOnPinInputShiftPinInput
Decrement counter on Pin input (both edges), Shift clock equals Pin input.

enumerator kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput
Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

enum _flexio_timer_reset_condition
Define type of timer reset condition.
Values:

enumerator kFLEXIO_TimerResetNever
Timer never reset.

enumerator kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput
Timer reset on Timer Pin equal to Timer Output.

enumerator kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput
Timer reset on Timer Trigger equal to Timer Output.

enumerator kFLEXIO_TimerResetOnTimerPinRisingEdge
Timer reset on Timer Pin rising edge.

enumerator kFLEXIO_TimerResetOnTimerTriggerRisingEdge
Timer reset on Trigger rising edge.

enumerator kFLEXIO_TimerResetOnTimerTriggerBothEdge
Timer reset on Trigger rising or falling edge.

enum _flexio_timer_disable_condition
Define type of timer disable condition.
Values:

enumerator kFLEXIO_TimerDisableNever
Timer never disabled.

enumerator kFLEXIO_TimerDisableOnPreTimerDisable
Timer disabled on Timer N-1 disable.

enumerator kFLEXIO_TimerDisableOnTimerCompare
Timer disabled on Timer compare.

enumerator kFLEXIO_TimerDisableOnTimerCompareTriggerLow
Timer disabled on Timer compare and Trigger Low.

enumerator kFLEXIO_TimerDisableOnPinBothEdge
Timer disabled on Pin rising or falling edge.

enumerator kFLEXIO_TimerDisableOnPinBothEdgeTriggerHigh
Timer disabled on Pin rising or falling edge provided Trigger is high.

enumerator kFLEXIO_TimerDisableOnTriggerFallingEdge
Timer disabled on Trigger falling edge.

enum _flexio_timer_enable_condition
Define type of timer enable condition.
Values:

enumerator kFLEXIO_TimerEnabledAlways
Timer always enabled.

enumerator kFLEXIO_TimerEnableOnPrevTimerEnable
Timer enabled on Timer N-1 enable.

enumerator kFLEXIO_TimerEnableOnTriggerHigh
Timer enabled on Trigger high.

enumerator kFLEXIO_TimerEnableOnTriggerHighPinHigh
Timer enabled on Trigger high and Pin high.

enumerator kFLEXIO_TimerEnableOnPinRisingEdge
Timer enabled on Pin rising edge.

enumerator kFLEXIO_TimerEnableOnPinRisingEdgeTriggerHigh
Timer enabled on Pin rising edge and Trigger high.

enumerator kFLEXIO_TimerEnableOnTriggerRisingEdge
Timer enabled on Trigger rising edge.

enumerator kFLEXIO_TimerEnableOnTriggerBothEdge
Timer enabled on Trigger rising or falling edge.

enum _flexio_timer_stop_bit_condition
Define type of timer stop bit generate condition.
Values:

enumerator kFLEXIO_TimerStopBitDisabled
Stop bit disabled.

enumerator kFLEXIO_TimerStopBitEnableOnTimerCompare
Stop bit is enabled on timer compare.

enumerator kFLEXIO_TimerStopBitEnableOnTimerDisable
Stop bit is enabled on timer disable.

enumerator kFLEXIO_TimerStopBitEnableOnTimerCompareDisable
Stop bit is enabled on timer compare and timer disable.

enum _flexio_timer_start_bit_condition
Define type of timer start bit generate condition.
Values:

enumerator kFLEXIO_TimerStartBitDisabled
Start bit disabled.

enumerator kFLEXIO_TimerStartBitEnabled
Start bit enabled.

enum _flexio_timer_output_state
FlexIO as PWM channel output state.
Values:

enumerator kFLEXIO_PwmLow
The output state of PWM channel is low

enumerator kFLEXIO_PwmHigh
The output state of PWM channel is high

enum _flexio_shifter_timer_polarity
Define type of timer polarity for shifter control.
Values:

enumerator kFLEXIO_ShifterTimerPolarityOnPositive
Shift on positive edge of shift clock.

enumerator kFLEXIO_ShifterTimerPolarityOnNegitive
Shift on negative edge of shift clock.

enum _flexio_shifter_mode
Define type of shifter working mode.
Values:

enumerator kFLEXIO_ShifterDisabled
Shifter is disabled.

enumerator kFLEXIO_ShifterModeReceive
Receive mode.

enumerator kFLEXIO_ShifterModeTransmit
Transmit mode.

enumerator kFLEXIO_ShifterModeMatchStore
Match store mode.

enumerator kFLEXIO_ShifterModeMatchContinuous
Match continuous mode.

enumerator kFLEXIO_ShifterModeState
SHIFTBUF contents are used for storing programmable state attributes.

enumerator kFLEXIO_ShifterModeLogic
SHIFTBUF contents are used for implementing programmable logic look up table.

enum _flexio_shifter_input_source
Define type of shifter input source.
Values:

enumerator kFLEXIO_ShifterInputFromPin
Shifter input from pin.

enumerator kFLEXIO_ShifterInputFromNextShifterOutput
Shifter input from Shifter N+1.

enum _flexio_shifter_stop_bit
Define of STOP bit configuration.
Values:

enumerator kFLEXIO_ShifterStopBitDisable
Disable shifter stop bit.

enumerator kFLEXIO_ShifterStopBitLow
Set shifter stop bit to logic low level.

enumerator kFLEXIO_ShifterStopBitHigh
Set shifter stop bit to logic high level.

enum _flexio_shifter_start_bit
Define type of START bit configuration.
Values:

enumerator kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable
Disable shifter start bit, transmitter loads data on enable.

enumerator kFLEXIO_ShifterStartBitDisabledLoadDataOnShift
Disable shifter start bit, transmitter loads data on first shift.

enumerator kFLEXIO_ShifterStartBitLow
Set shifter start bit to logic low level.

enumerator kFLEXIO_ShifterStartBitHigh
Set shifter start bit to logic high level.

enum _flexio_shifter_buffer_type
Define FlexIO shifter buffer type.
Values:

enumerator kFLEXIO_ShifterBuffer
Shifter Buffer N Register.

enumerator kFLEXIO_ShifterBufferBitSwapped
Shifter Buffer N Bit Byte Swapped Register.

enumerator kFLEXIO_ShifterBufferByteSwapped
Shifter Buffer N Byte Swapped Register.

enumerator kFLEXIO_ShifterBufferBitByteSwapped
Shifter Buffer N Bit Swapped Register.

enumerator kFLEXIO_ShifterBufferNibbleByteSwapped
Shifter Buffer N Nibble Byte Swapped Register.

enumerator kFLEXIO_ShifterBufferHalfWordSwapped
Shifter Buffer N Half Word Swapped Register.

enumerator kFLEXIO_ShifterBufferNibbleSwapped
Shifter Buffer N Nibble Swapped Register.

enum _flexio_gpio_direction
FLEXIO gpio direction definition.
Values:

enumerator kFLEXIO_DigitalInput
Set current pin as digital input

enumerator kFLEXIO_DigitalOutput
Set current pin as digital output

enum _flexio_pin_input_config
FLEXIO gpio input config.
Values:

enumerator kFLEXIO_InputInterruptDisabled
Interrupt request is disabled.

enumerator kFLEXIO_InputInterruptEnable
Interrupt request is enable.

enumerator kFLEXIO_FlagRisingEdgeEnable
Input pin flag on rising edge.

enumerator kFLEXIO_FlagFallingEdgeEnable
Input pin flag on falling edge.

typedef enum *_flexio_timer_trigger_polarity* flexio_timer_trigger_polarity_t
Define time of timer trigger polarity.

typedef enum *_flexio_timer_trigger_source* flexio_timer_trigger_source_t
Define type of timer trigger source.

typedef enum *_flexio_pin_config* flexio_pin_config_t
Define type of timer/shifter pin configuration.

typedef enum *_flexio_pin_polarity* flexio_pin_polarity_t
Definition of pin polarity.

typedef enum *_flexio_timer_mode* flexio_timer_mode_t
Define type of timer work mode.

typedef enum *_flexio_timer_output* flexio_timer_output_t
Define type of timer initial output or timer reset condition.

typedef enum *_flexio_timer_decrement_source* flexio_timer_decrement_source_t
Define type of timer decrement.

typedef enum *_flexio_timer_reset_condition* flexio_timer_reset_condition_t
Define type of timer reset condition.

typedef enum *_flexio_timer_disable_condition* flexio_timer_disable_condition_t
Define type of timer disable condition.

typedef enum *_flexio_timer_enable_condition* flexio_timer_enable_condition_t
Define type of timer enable condition.

typedef enum *_flexio_timer_stop_bit_condition* flexio_timer_stop_bit_condition_t
Define type of timer stop bit generate condition.

typedef enum *_flexio_timer_start_bit_condition* flexio_timer_start_bit_condition_t
Define type of timer start bit generate condition.

typedef enum *_flexio_timer_output_state* flexio_timer_output_state_t

FlexIO as PWM channel output state.

typedef enum *_flexio_shifter_timer_polarity* flexio_shifter_timer_polarity_t

Define type of timer polarity for shifter control.

typedef enum *_flexio_shifter_mode* flexio_shifter_mode_t

Define type of shifter working mode.

typedef enum *_flexio_shifter_input_source* flexio_shifter_input_source_t

Define type of shifter input source.

typedef enum *_flexio_shifter_stop_bit* flexio_shifter_stop_bit_t

Define of STOP bit configuration.

typedef enum *_flexio_shifter_start_bit* flexio_shifter_start_bit_t

Define type of START bit configuration.

typedef enum *_flexio_shifter_buffer_type* flexio_shifter_buffer_type_t

Define FlexIO shifter buffer type.

typedef struct *_flexio_config_* flexio_config_t

Define FlexIO user configuration structure.

typedef struct *_flexio_timer_config* flexio_timer_config_t

Define FlexIO timer configuration structure.

typedef struct *_flexio_shifter_config* flexio_shifter_config_t

Define FlexIO shifter configuration structure.

typedef enum *_flexio_gpio_direction* flexio_gpio_direction_t

FLEXIO gpio direction definition.

typedef enum *_flexio_pin_input_config* flexio_pin_input_config_t

FLEXIO gpio input config.

typedef struct *_flexio_gpio_config* flexio_gpio_config_t

The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

typedef void (*flexio_isr_t)(void *base, void *handle)

typedef for FlexIO simulated driver interrupt handler.

FLEXIO_Type *const s_flexioBases[]

Pointers to flexio bases for each instance.

const *clock_ip_name_t* s_flexioClocks[]

Pointers to flexio clocks for each instance.

void FLEXIO_SetPinConfig(FLEXIO_Type *base, uint32_t pin, *flexio_gpio_config_t* *config)

Configure a FLEXIO pin used by the board.

To Config the FLEXIO PIN, define a pin configuration, as either input or output, in the user file. Then, call the *FLEXIO_SetPinConfig()* function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
flexio_gpio_config_t config =
{
  kFLEXIO_DigitalInput,
  0U,
  kFLEXIO_FlagRisingEdgeEnable | kFLEXIO_InputInterruptEnable,
}
Define a digital output pin configuration,
flexio_gpio_config_t config =
{
  kFLEXIO_DigitalOutput,
  0U,
  0U
}
```

Parameters

- base – FlexIO peripheral base address
- pin – FLEXIO pin number.
- config – FLEXIO pin configuration pointer.

FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x)

Calculate FlexIO timer trigger.

FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(x)

FLEXIO_TIMER_TRIGGER_SEL_TIMn(x)

struct _flexio_config_

#include <fsl_flexio.h> Define FlexIO user configuration structure.

Public Members

bool enableFlexio

Enable/disable FlexIO module

bool enableInDoze

Enable/disable FlexIO operation in doze mode

bool enableInDebug

Enable/disable FlexIO operation in debug mode

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

struct _flexio_timer_config

#include <fsl_flexio.h> Define FlexIO timer configuration structure.

Public Members

uint32_t triggerSelect

The internal trigger selection number using MACROs.

flexio_timer_trigger_polarity_t triggerPolarity

Trigger Polarity.

flexio_timer_trigger_source_t triggerSource

Trigger Source, internal (see 'trgsel') or external.

flexio_pin_config_t pinConfig

Timer Pin Configuration.

uint32_t pinSelect

Timer Pin number Select.

flexio_pin_polarity_t pinPolarity

Timer Pin Polarity.

flexio_timer_mode_t timerMode

Timer work Mode.

flexio_timer_output_t timerOutput

Configures the initial state of the Timer Output and whether it is affected by the Timer reset.

flexio_timer_decrement_source_t timerDecrement

Configures the source of the Timer decrement and the source of the Shift clock.

flexio_timer_reset_condition_t timerReset

Configures the condition that causes the timer counter (and optionally the timer output) to be reset.

flexio_timer_disable_condition_t timerDisable

Configures the condition that causes the Timer to be disabled and stop decrementing.

flexio_timer_enable_condition_t timerEnable

Configures the condition that causes the Timer to be enabled and start decrementing.

flexio_timer_stop_bit_condition_t timerStop
: Timer STOP Bit generation.

flexio_timer_start_bit_condition_t timerStart
: Timer STRAT Bit generation.

uint32_t timerCompare
: Value for Timer Compare N Register.

struct __flexio__shifter__config
: *#include <fsl_flexio.h>* Define FlexIO shifter configuration structure.

Public Members

uint32_t timerSelect
: Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.

flexio_shifter_timer_polarity_t timerPolarity
: Timer Polarity.

flexio_pin_config_t pinConfig
: Shifter Pin Configuration.

uint32_t pinSelect
: Shifter Pin number Select.

flexio_pin_polarity_t pinPolarity
: Shifter Pin Polarity.

flexio_shifter_mode_t shifterMode
: Configures the mode of the Shifter.

uint32_t parallelWidth
: Configures the parallel width when using parallel mode.

flexio_shifter_input_source_t inputSource
: Selects the input source for the shifter.

flexio_shifter_stop_bit_t shifterStop
: Shifter STOP bit.

flexio_shifter_start_bit_t shifterStart
: Shifter START bit.

struct __flexio__gpio__config
: *#include <fsl_flexio.h>* The FLEXIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, use inputConfig param. If configured as an output pin, use outputLogic.

Public Members

flexio_gpio_direction_t pinDirection

FLEXIO pin direction, input or output

uint8_t outputLogic

Set a default output logic, which has no use in input

uint8_t inputConfig

Set an input config

2.16 FlexIO eDMA I2S Driver

void FLEXIO_I2S_TransferTxCreateHandleEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the FlexIO I2S eDMA handle.

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.
- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

void FLEXIO_I2S_TransferRxCreateHandleEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the FlexIO I2S Rx eDMA handle.

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer.
- callback – FlexIO I2S eDMA callback function called while finished a block.
- userData – User parameter for callback.

- dmaHandle – eDMA handle for FlexIO I2S. This handle is a static value allocated by users.

void FLEXIO_I2S_TransferSetFormatEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_format_t* *format, uint32_t srcClock_Hz)

Configures the FlexIO I2S Tx audio format.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S eDMA handle pointer
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode.

status_t FLEXIO_I2S_TransferSendEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs a non-blocking FlexIO I2S transfer using DMA.

Note: This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a FlexIO I2S eDMA send successfully.
- kStatus_InvalidArgument – The input arguments is invalid.
- kStatus_TxBusy – FlexIO I2S is busy sending data.

status_t FLEXIO_I2S_TransferReceiveEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs a non-blocking FlexIO I2S receive using eDMA.

Note: This interface returned immediately after transfer initiates. Users should call FLEXIO_I2S_GetReceiveRemainingBytes to poll the transfer status and check whether the FlexIO I2S transfer is finished.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a FlexIO I2S eDMA receive successfully.
- kStatus_InvalidArgument – The input arguments is invalid.
- kStatus_RxBusy – FlexIO I2S is busy receiving data.

void FLEXIO_I2S_TransferAbortSendEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle)

Aborts a FlexIO I2S transfer using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

void FLEXIO_I2S_TransferAbortReceiveEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle)

Aborts a FlexIO I2S receive using eDMA.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.

status_t FLEXIO_I2S_TransferGetSendCountEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, size_t *count)

Gets the remaining bytes to be sent.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes sent.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

status_t FLEXIO_I2S_TransferGetReceiveCountEDMA(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, size_t *count)

Get the remaining bytes to be received.

Parameters

- base – FlexIO I2S peripheral base address.
- handle – FlexIO I2S DMA handle pointer.
- count – Bytes received.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION

FlexIO I2S EDMA driver version 2.1.9.

typedef struct *_flexio_i2s_edma_handle* flexio_i2s_edma_handle_t

typedef void (*flexio_i2s_edma_callback_t)(*FLEXIO_I2S_Type* *base, *flexio_i2s_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO I2S eDMA transfer callback function for finish and error.

struct _flexio_i2s_edma_handle

#include <fsl_flexio_i2s_edma.h> FlexIO I2S DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for FlexIO I2S send

uint8_t bytesPerFrame

Bytes in a frame

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint32_t state

Internal state for FlexIO I2S eDMA transfer

flexio_i2s_edma_callback_t callback

Callback for users while transfer finish or error occurred

void *userData

User callback parameter

edma_tcd_t tcd[(4U) + 1U]

TCD pool for eDMA transfer.

flexio_i2s_transfer_t queue[(4U)]

Transfer queue storing queued transfer.

size_t transferSize[(4U)]

Data bytes need to transfer

volatile uint8_t queueUser

Index for user to queue transfer.

volatile uint8_t queueDriver

Index for driver to get the transfer data and size

2.17 FlexIO eDMA MCU Interface LCD Driver

status_t FLEXIO_MCULCD_TransferCreateHandleEDMA(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_edma_handle_t* *handle, *flexio_mculcd_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txDmaHandle, *edma_handle_t* *rxDmaHandle)

Initializes the FLEXO MCULCD master eDMA handle.

This function initializes the FLEXO MCULCD master eDMA handle which can be used for other FLEXO MCULCD transactional APIs. For a specified FLEXO MCULCD instance, call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_MCULCD_Type structure.
- handle – Pointer to flexio_mculcd_edma_handle_t structure to store the transfer state.
- callback – MCULCD transfer complete callback, NULL means no callback.
- userData – callback function parameter.
- txDmaHandle – User requested eDMA handle for FlexIO MCULCD eDMA TX, the DMA request source of this handle should be the first of TX shifters.
- rxDmaHandle – User requested eDMA handle for FlexIO MCULCD eDMA RX, the DMA request source of this handle should be the last of RX shifters.

Return values

kStatus_Success – Successfully create the handle.

status_t FLEXIO_MCULCD_TransferEDMA(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_edma_handle_t* *handle, *flexio_mculcd_transfer_t* *xfer)

Performs a non-blocking FlexIO MCULCD transfer using eDMA.

This function returns immediately after transfer initiates. To check whether the transfer is completed, user could:

a. Use the transfer completed callback;

b. Polling function FLEXIO_MCULCD_GetTransferCountEDMA

Parameters

- base – pointer to FLEXIO_MCULCD_Type structure.
- handle – pointer to flexio_mculcd_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO MCULCD transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_MCULCD_Busy – FlexIO MCULCD is not idle, it is running another transfer.

void FLEXIO_MCULCD_TransferAbortEDMA(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_edma_handle_t* *handle)

Aborts a FlexIO MCULCD transfer using eDMA.

Parameters

- base – pointer to FLEXIO_MCULCD_Type structure.
- handle – FlexIO MCULCD eDMA handle pointer.

status_t FLEXIO_MCULCD_TransferGetCountEDMA(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_edma_handle_t* *handle, size_t *count)

Gets the remaining bytes for FlexIO MCULCD eDMA transfer.

Parameters

- base – pointer to FLEXIO_MCULCD_Type structure.
- handle – FlexIO MCULCD eDMA handle pointer.
- count – Number of count transferred so far by the eDMA transaction.

Return values

- kStatus_Success – Get the transferred count Successfully.
- kStatus_NoTransferInProgress – No transfer in process.

typedef struct *_flexio_mculcd_edma_handle* flexio_mculcd_edma_handle_t

typedef for flexio_mculcd_edma_handle_t in advance.

typedef void (*flexio_mculcd_edma_transfer_callback_t)(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO MCULCD master callback for transfer complete.

When transfer finished, the callback function is called and returns the status as kStatus_FLEXIO_MCULCD_Idle.

FSL_FLEXIO_MCULCD_EDMA_DRIVER_VERSION

FlexIO MCULCD EDMA driver version.

struct _flexio_mculcd_edma_handle

#include <fsl_flexio_mculcd_edma.h> FlexIO MCULCD eDMA transfer handle, users should not touch the content of the handle.

Public Members

FLEXIO_MCULCD_Type *base

Pointer to the FLEXIO_MCULCD_Type.

uint8_t txShifterNum

Number of shifters used for TX.

uint8_t rxShifterNum

Number of shifters used for RX.

uint32_t minorLoopBytes

eDMA transfer minor loop bytes.

edma_modulo_t txEdmaModulo

Modulo value for the FlexIO shifter buffer access.

edma_modulo_t rxEdmaModulo

Modulo value for the FlexIO shifter buffer access.

uint32_t dataAddrOrSameValue

When sending the same value for many times, this is the value to send. When writing or reading array, this is the address of the data array.

size_t dataCount

Total count to be transferred.

volatile size_t remainingCount

Remaining count still not transfered.

volatile uint32_t state

FlexIO MCULCD driver internal state.

edma_handle_t *txDmaHandle

DMA handle for MCULCD TX

edma_handle_t *rxDmaHandle

DMA handle for MCULCD RX

flexio_mculcd_edma_transfer_callback_t completionCallback

Callback for MCULCD DMA transfer

void *userData

User Data for MCULCD DMA callback

2.18 FlexIO eDMA SPI Driver

status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, *flexio_spi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txHandle, *edma_handle_t* *rxHandle)

Initializes the FlexIO SPI master eDMA handle.

This function initializes the FlexIO SPI master eDMA handle which can be used for other FlexIO SPI master transactional APIs. For a specified FlexIO SPI instance, call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

status_t FLEXIO_SPI_MasterTransferEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_MasterGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

void FLEXIO_SPI_MasterTransferAbortEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle)

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.

status_t FLEXIO_SPI_MasterTransferGetCountEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, size_t *count)

Gets the number of bytes transferred so far using FlexIO SPI master eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

static inline void FLEXIO_SPI_SlaveTransferCreateHandleEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, *flexio_spi_slave_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txHandle, *edma_handle_t* *rxHandle)

Initializes the FlexIO SPI slave eDMA handle.

This function initializes the FlexIO SPI slave eDMA handle.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- callback – SPI callback, NULL means no callback.
- userData – callback function parameter.
- txHandle – User requested eDMA handle for FlexIO SPI TX eDMA transfer.
- rxHandle – User requested eDMA handle for FlexIO SPI RX eDMA transfer.

status_t FLEXIO_SPI_SlaveTransferEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Performs a non-blocking FlexIO SPI transfer using eDMA.

Note: This interface returns immediately after transfer initiates. Call FLEXIO_SPI_SlaveGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
- xfer – Pointer to FlexIO SPI transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – FlexIO SPI is not idle, is running another transfer.

static inline void FLEXIO_SPI_SlaveTransferAbortEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle)

Aborts a FlexIO SPI transfer using eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.

static inline *status_t* FLEXIO_SPI_SlaveTransferGetCountEDMA(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, size_t *count)

Gets the number of bytes transferred so far using FlexIO SPI slave eDMA.

Parameters

- base – Pointer to FLEXIO_SPI_Type structure.
- handle – FlexIO SPI eDMA handle pointer.
- count – Number of bytes transferred so far by the non-blocking transaction.

FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION

FlexIO SPI EDMA driver version.

typedef struct *_flexio_spi_master_edma_handle* flexio_spi_master_edma_handle_t

typedef for flexio_spi_master_edma_handle_t in advance.

typedef *flexio_spi_master_edma_handle_t* flexio_spi_slave_edma_handle_t

Slave handle is the same with master handle.

typedef void (*flexio_spi_master_edma_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_master_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO SPI master callback for finished transmit.

typedef void (*flexio_spi_slave_edma_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_edma_handle_t* *handle, *status_t* status, void *userData)

FlexIO SPI slave callback for finished transmit.

struct _flexio_spi_master_edma_handle

#include <fsl_flexio_spi_edma.h> FlexIO SPI eDMA transfer handle, users should not touch the content of the handle.

Public Members

size_t transferSize

Total bytes to be transferred.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

bool txInProgress

Send transfer in progress

bool rxInProgress

Receive transfer in progress

edma_handle_t *txHandle

DMA handler for SPI send

edma_handle_t *rxHandle

DMA handler for SPI receive

flexio_spi_master_edma_transfer_callback_t callback

Callback for SPI DMA transfer

void *userData

User Data for SPI DMA callback

2.19 FlexIO eDMA UART Driver

status_t FLEXIO_UART_TransferCreateHandleEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *flexio_uart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle)

Initializes the UART handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_UART_Type.
- handle – Pointer to flexio_uart_edma_handle_t structure.
- callback – The callback function.
- userData – The parameter of the callback function.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO SPI eDMA type/handle table out of range.

status_t FLEXIO_UART_TransferSendEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – UART handle pointer.
- xfer – UART eDMA transfer structure, see *flexio_uart_transfer_t*.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_FLEXIO_UART_TxBusy – Previous transfer on going.

status_t FLEXIO_UART_TransferReceiveEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- xfer – UART eDMA transfer structure, see *flexio_uart_transfer_t*.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_UART_RxBusy – Previous transfer on going.

void FLEXIO_UART_TransferAbortSendEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle)

Aborts the sent data which using eDMA.

This function aborts sent data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

void FLEXIO_UART_TransferAbortReceiveEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle)

Aborts the receive data which using eDMA.

This function aborts the receive data which using eDMA.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure

status_t FLEXIO_UART_TransferGetSendCountEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, size_t *count)

Gets the number of bytes sent out.

This function gets the number of bytes sent out.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

status_t FLEXIO_UART_TransferGetReceiveCountEDMA(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, size_t *count)

Gets the number of bytes received.

This function gets the number of bytes received.

Parameters

- base – Pointer to FLEXIO_UART_Type
- handle – Pointer to flexio_uart_edma_handle_t structure
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

FSL_FLEXIO_UART_EDMA_DRIVER_VERSION

FlexIO UART EDMA driver version.

typedef struct *_flexio_uart_edma_handle* flexio_uart_edma_handle_t

typedef void (*flexio_uart_edma_transfer_callback_t)(*FLEXIO_UART_Type* *base, *flexio_uart_edma_handle_t* *handle, *status_t* status, void *userData)

UART transfer callback function.

struct _flexio_uart_edma_handle

#include <fsl_flexio_uart_edma.h> UART eDMA handle.

Public Members

flexio_uart_edma_transfer_callback_t callback

Callback function.

void *userData

UART callback function parameter.

size_t txDataSizeAll

Total bytes to be sent.

size_t rxDataSizeAll

Total bytes to be received.

edma_handle_t *txEdmaHandle

The eDMA TX channel used.

edma_handle_t *rxEdmaHandle

The eDMA RX channel used.

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

volatile uint8_t txState
: TX transfer state.

volatile uint8_t rxState
: RX transfer state

2.20 FlexIO I2C Master Driver

status_t FLEXIO_I2C_CheckForBusyBus(*FLEXIO_I2C_Type* *base)

Make sure the bus isn't already pulled down.

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure..

Return values

- kStatus_Success –
- kStatus_FLEXIO_I2C_Busy –

status_t FLEXIO_I2C_MasterInit(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.

Example

```
FLEXIO_I2C_Type base = {
.flexioBase = FLEXIO,
.SDAPinIndex = 0,
.SCLPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- masterConfig – Pointer to flexio_i2c_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Initialization successful
- kStatus_InvalidArgument – The source clock exceed upper range limitation

void FLEXIO_I2C_MasterDeinit(*FLEXIO_I2C_Type* *base)

De-initializes the FlexIO I2C master peripheral. Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterGetDefaultConfig(*flexio_i2c_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO module. The configuration can be used directly for calling the *FLEXIO_I2C_MasterInit()*.

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – Pointer to flexio_i2c_master_config_t structure.

static inline void FLEXIO_I2C_MasterEnable(*FLEXIO_I2C_Type* *base, bool enable)

Enables/disables the FlexIO module operation.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – Pass true to enable module, false does not have any effect.

uint32_t FLEXIO_I2C_MasterGetStatusFlags(*FLEXIO_I2C_Type* *base)

Gets the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure

Returns

Status flag, use status flag to AND *_flexio_i2c_master_status_flags* can get the related status.

void FLEXIO_I2C_MasterClearStatusFlags(*FLEXIO_I2C_Type* *base, uint32_t mask)

Clears the FlexIO I2C master status flags.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO_I2C_RxFullFlag
  - kFLEXIO_I2C_ReceiveNakFlag

void FLEXIO_I2C_MasterEnableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Enables the FlexIO i2c master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source. Currently only one interrupt request source:
  - kFLEXIO_I2C_TransferCompleteInterruptEnable

void FLEXIO_I2C_MasterDisableInterrupts(*FLEXIO_I2C_Type* *base, uint32_t mask)

Disables the FlexIO I2C master interrupt requests.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- mask – Interrupt source.

void FLEXIO_I2C_MasterSetBaudRate(*FLEXIO_I2C_Type* *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the FlexIO I2C master transfer baudrate.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- baudRate_Bps – the baud rate value in HZ
- srcClock_Hz – source clock in HZ

void FLEXIO_I2C_MasterStart(*FLEXIO_I2C_Type* *base, uint8_t address, *flexio_i2c_direction_t* direction)

Sends START + 7-bit address to the bus.

Note: This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO_I2C_RxFullFlag status is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- address – 7-bit address.
- direction – transfer direction. This parameter is one of the values in flexio_i2c_direction_t:
  - kFLEXIO_I2C_Write: Transmit
  - kFLEXIO_I2C_Read: Receive

void FLEXIO_I2C_MasterStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterRepeatedStart(*FLEXIO_I2C_Type* *base)

Sends the repeated start signal on the bus.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterAbortStop(*FLEXIO_I2C_Type* *base)

Sends the stop signal when transfer is still on-going.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

void FLEXIO_I2C_MasterEnableAck(*FLEXIO_I2C_Type* *base, bool enable)

Configures the sent ACK/NAK for the following byte.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- enable – True to configure send ACK, false configure to send NAK.

status_t FLEXIO_I2C_MasterSetTransferCount(*FLEXIO_I2C_Type* *base, uint16_t count)

Sets the number of bytes to be transferred from a start signal to a stop signal.

Note: Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- count – Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

- kStatus_Success – Successfully configured the count.
- kStatus_InvalidArgument – Input argument is invalid.

static inline void FLEXIO_I2C_MasterWriteByte(*FLEXIO_I2C_Type* *base, uint32_t data)

Writes one byte of data to the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- data – a byte of data.

static inline uint8_t FLEXIO_I2C_MasterReadByte(*FLEXIO_I2C_Type* *base)

Reads one byte of data from the I2C bus.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.

Returns

data byte read.

status_t FLEXIO_I2C_MasterWriteBlocking(*FLEXIO_I2C_Type* *base, const uint8_t *txBuff, uint8_t txSize)

Sends a buffer of data in bytes.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- txBuff – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Nak – Receive NAK during writing data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterReadBlocking(*FLEXIO_I2C_Type* *base, uint8_t *rxBuff, uint8_t rxSize)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- rxBuff – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully read data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

status_t FLEXIO_I2C_MasterTransferBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_transfer_t* *xfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

- base – pointer to FLEXIO_I2C_Type structure.
- xfer – pointer to flexio_i2c_master_transfer_t structure.

Returns

status of status_t.

status_t FLEXIO_I2C_MasterTransferCreateHandle(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *flexio_i2c_master_transfer_callback_t* callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure to store the transfer state.
- callback – Pointer to user callback function.
- userData – User param passed to the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/isr table out of range.

status_t FLEXIO_I2C_MasterTransferNonBlocking(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *flexio_i2c_master_transfer_t* *xfer)

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: The API returns immediately after the transfer initiates. Call FLEXIO_I2C_MasterTransferGetCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXIO_I2C_Busy, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state
- xfer – pointer to flexio_i2c_master_transfer_t structure

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_FLEXIO_I2C_Busy – FlexIO I2C is not idle, is running another transfer.

status_t FLEXIO_I2C_MasterTransferGetCount(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, size_t *count)

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure.
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.
- kStatus_Success – Successfully return the count.

void FLEXIO_I2C_MasterTransferAbort(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle)

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – Pointer to FLEXIO_I2C_Type structure
- handle – Pointer to flexio_i2c_master_handle_t structure which stores the transfer state

void FLEXIO_I2C_MasterTransferHandleIRQ(void *i2cType, void *i2cHandle)

Master interrupt handler.

Parameters

- i2cType – Pointer to FLEXIO_I2C_Type structure
- i2cHandle – Pointer to flexio_i2c_master_transfer_t structure

FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION

FlexIO I2C transfer status.

Values:

enumerator kStatus_FLEXIO_I2C_Busy

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Idle

I2C is busy doing transfer.

enumerator kStatus_FLEXIO_I2C_Nak

NAK received during transfer.

enumerator kStatus_FLEXIO_I2C_Timeout

Timeout polling status flags.

enum _flexio_i2c_master_interrupt

Define FlexIO I2C master interrupt mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyInterruptEnable

Tx buffer empty interrupt enable.

enumerator kFLEXIO_I2C_RxFullInterruptEnable

Rx buffer full interrupt enable.

enum _flexio_i2c_master_status_flags

Define FlexIO I2C master status mask.

Values:

enumerator kFLEXIO_I2C_TxEmptyFlag

Tx shifter empty flag.

enumerator kFLEXIO_I2C_RxFullFlag

Rx shifter full/Transfer complete flag.

enumerator kFLEXIO_I2C_ReceiveNakFlag

Receive NAK flag.

enum _flexio_i2c_direction

Direction of master transfer.

Values:

enumerator kFLEXIO_I2C_Write

Master send to slave.

enumerator kFLEXIO_I2C_Read

Master receive from slave.

typedef enum *_flexio_i2c_direction* flexio_i2c_direction_t

Direction of master transfer.

typedef struct *_flexio_i2c_type* FLEXIO_I2C_Type

Define FlexIO I2C master access structure typedef.

typedef struct *_flexio_i2c_master_config* flexio_i2c_master_config_t

Define FlexIO I2C master user configuration structure.

typedef struct *_flexio_i2c_master_transfer* flexio_i2c_master_transfer_t

Define FlexIO I2C master transfer structure.

typedef struct *_flexio_i2c_master_handle* flexio_i2c_master_handle_t

FlexIO I2C master handle typedef.

typedef void (*flexio_i2c_master_transfer_callback_t)(*FLEXIO_I2C_Type* *base, *flexio_i2c_master_handle_t* *handle, *status_t* status, void *userData)

FlexIO I2C master transfer callback typedef.

I2C_RETRY_TIMES

Retry times for waiting flag.

struct _flexio_i2c_type

#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master access structure typedef.

Public Members

FLEXIO_Type *flexioBase
: FlexIO base pointer.

uint8_t SDAPinIndex
: Pin select for I2C SDA.

uint8_t SCLPinIndex
: Pin select for I2C SCL.

uint8_t shifterIndex[2]
: Shifter index used in FlexIO I2C.

uint8_t timerIndex[3]
: Timer index used in FlexIO I2C.

uint32_t baudrate
: Master transfer baudrate, used to calculate delay time.

struct __flexio_i2c_master_config
: *#include <fsl_flexio_i2c_master.h>* Define FlexIO I2C master user configuration structure.

Public Members

bool enableMaster
: Enables the FlexIO I2C peripheral at initialization time.

bool enableInDoze
: Enable/disable FlexIO operation in doze mode.

bool enableInDebug
: Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
: Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
: Baud rate in Bps.

struct __flexio_i2c_master_transfer
: *#include <fsl_flexio_i2c_master.h>* Define FlexIO I2C master transfer structure.

Public Members

uint32_t flags
Transfer flag which controls the transfer, reserved for FlexIO I2C.

uint8_t slaveAddress
7-bit slave address.

flexio_i2c_direction_t direction
Transfer direction, read or write.

uint32_t subaddress
Sub address. Transferred MSB first.

uint8_t subaddressSize
Size of sub address.

uint8_t volatile *data
Transfer buffer.

volatile size_t dataSize
Transfer size.

struct __flexio_i2c_master_handle
#include <fsl_flexio_i2c_master.h> Define FlexIO I2C master handle structure.

Public Members

flexio_i2c_master_transfer_t transfer
FlexIO I2C master transfer copy.

size_t transferSize
Total bytes to be transferred.

uint8_t state
Transfer state maintained during transfer.

flexio_i2c_master_transfer_callback_t completionCallback
Callback function called at transfer event. Callback function called at transfer event.

void *userData
Callback parameter passed to callback function.

bool needRestart
Whether master needs to send re-start signal.

2.21 FlexIO I2S Driver

void FLEXIO_I2S_Init(*FLEXIO_I2S_Type* *base, const *flexio_i2s_config_t* *config)

Initializes the FlexIO I2S.

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by *FLEXIO_I2S_GetDefaultConfig()*.

Note: This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

Parameters

- base – FlexIO I2S base pointer
- config – FlexIO I2S configure structure.

void FLEXIO_I2S_GetDefaultConfig(*flexio_i2s_config_t* *config)

Sets the FlexIO I2S configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in *FLEXIO_I2S_Init()*. Users may use the initialized structure unchanged in *FLEXIO_I2S_Init()* or modify some fields of the structure before calling *FLEXIO_I2S_Init()*.

Parameters

- config – pointer to master configuration structure

void FLEXIO_I2S_Deinit(*FLEXIO_I2S_Type* *base)

De-initializes the FlexIO I2S.

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the FLEXO_I2S_Init to use the FlexIO I2S module.

Parameters

- base – FlexIO I2S base pointer

static inline void FLEXIO_I2S_Enable(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S module operation.

Parameters

- base – Pointer to FLEXIO_I2S_Type
- enable – True to enable, false dose not have any effect.

uint32_t FLEXIO_I2S_GetStatusFlags(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S status flags.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

Status flag, which are ORed by the enumerators in the _flexio_i2s_status_flags.

void FLEXIO_I2S_EnableInterrupts(*FLEXIO_I2S_Type* *base, uint32_t mask)

Enables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- mask – interrupt source

void FLEXIO_I2S_DisableInterrupts(*FLEXIO_I2S_Type* *base, uint32_t mask)

Disables the FlexIO I2S interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – pointer to FLEXIO_I2S_Type structure
- mask – interrupt source

static inline void FLEXIO_I2S_TxEnableDMA(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S Tx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

static inline void FLEXIO_I2S_RxEnableDMA(*FLEXIO_I2S_Type* *base, bool enable)

Enables/disables the FlexIO I2S Rx DMA requests.

Parameters

- base – FlexIO I2S base pointer
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t FLEXIO_I2S_TxGetDataRegisterAddress(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S send data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

FlexIO i2s send data register address.

static inline uint32_t FLEXIO_I2S_RxGetDataRegisterAddress(*FLEXIO_I2S_Type* *base)

Gets the FlexIO I2S receive data register address.

This function returns the I2S data register address, mainly used by DMA/eDMA.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure

Returns

FlexIO i2s receive data register address.

void FLEXIO_I2S_MasterSetFormat(*FLEXIO_I2S_Type* *base, *flexio_i2s_format_t* *format, uint32_t srcClock_Hz)

Configures the FlexIO I2S audio format in master mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- format – Pointer to FlexIO I2S audio data format structure.
- srcClock_Hz – I2S master clock source frequency in Hz.

void FLEXIO_I2S_SlaveSetFormat(*FLEXIO_I2S_Type* *base, *flexio_i2s_format_t* *format)

Configures the FlexIO I2S audio format in slave mode.

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- format – Pointer to FlexIO I2S audio data format structure.

status_t FLEXIO_I2S_WriteBlocking(*FLEXIO_I2S_Type* *base, uint8_t bitWidth, uint8_t *txData, size_t size)

Sends data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- txData – Pointer to the data to be written.
- size – Bytes to be written.

Return values

- kStatus_Success – Successfully write data.
- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

static inline void FLEXIO_I2S_WriteData(*FLEXIO_I2S_Type* *base, uint8_t bitWidth, uint32_t data)

Writes data into a data register.

Parameters

- base – FlexIO I2S base pointer.
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- data – Data to be written.

status_t FLEXIO_I2S_ReadBlocking(*FLEXIO_I2S_Type* *base, uint8_t bitWidth, uint8_t *rxData, size_t size)

Receives a piece of data using a blocking method.

Note: This function blocks via polling until data is ready to be sent.

Parameters

- base – FlexIO I2S base pointer
- bitWidth – How many bits in a audio word, usually 8/16/24/32 bits.
- rxData – Pointer to the data to be read.
- size – Bytes to be read.

Return values

- kStatus_Success – Successfully read data.

- kStatus_FLEXIO_I2C_Timeout – Timeout polling status flags.

static inline uint32_t FLEXIO_I2S_ReadData(*FLEXIO_I2S_Type* *base)

Reads a data from the data register.

Parameters

- base – FlexIO I2S base pointer

Returns

Data read from data register.

void FLEXIO_I2S_TransferTxCreateHandle(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_callback_t* callback, void *userData)

Initializes the FlexIO I2S handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure
- handle – Pointer to flexio_i2s_handle_t structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

void FLEXIO_I2S_TransferSetFormat(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_format_t* *format, uint32_t srcClock_Hz)

Configures the FlexIO I2S audio format.

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – FlexIO I2S handle pointer.
- format – Pointer to audio data format structure.
- srcClock_Hz – FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode.

void FLEXIO_I2S_TransferRxCreateHandle(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_callback_t* callback, void *userData)

Initializes the FlexIO I2S receive handle.

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure to store the transfer state.
- callback – FlexIO I2S callback function, which is called while finished a block.
- userData – User parameter for the FlexIO I2S callback.

status_t FLEXIO_I2S_TransferSendNonBlocking(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- xfer – Pointer to flexio_i2s_transfer_t structure

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_FLEXIO_I2S_TxBusy – Previous transmission still not finished, data not all written to TX register yet.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t FLEXIO_I2S_TransferReceiveNonBlocking(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *flexio_i2s_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on FlexIO I2S.

Note: The API returns immediately after transfer initiates. Call FLEXIO_I2S_GetRemainingBytes to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- xfer – Pointer to flexio_i2s_transfer_t structure

Return values

- kStatus_Success – Successfully start the data receive.
- kStatus_FLEXIO_I2S_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

void FLEXIO_I2S_TransferAbortSend(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle)

Aborts the current send.

Note: This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.

- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state

void FLEXIO_I2S_TransferAbortReceive(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle)

Aborts the current receive.

Note: This API can be called at any time when interrupt non-blocking transfer initiates to abort the transfer in a early time.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state

status_t FLEXIO_I2S_TransferGetSendCount(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, size_t *count)

Gets the remaining bytes to be sent.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- count – Bytes sent.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

status_t FLEXIO_I2S_TransferGetReceiveCount(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, size_t *count)

Gets the remaining bytes to be received.

Parameters

- base – Pointer to FLEXIO_I2S_Type structure.
- handle – Pointer to flexio_i2s_handle_t structure which stores the transfer state
- count – Bytes recieved.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

Returns

count Bytes received.

void FLEXIO_I2S_TransferTxHandleIRQ(void *i2sBase, void *i2sHandle)

Tx interrupt handler.

Parameters

- i2sBase – Pointer to FLEXIO_I2S_Type structure.

- i2sHandle – Pointer to flexio_i2s_handle_t structure

void FLEXIO_I2S_TransferRxHandleIRQ(void *i2sBase, void *i2sHandle)

Rx interrupt handler.

Parameters

- i2sBase – Pointer to FLEXIO_I2S_Type structure.
- i2sHandle – Pointer to flexio_i2s_handle_t structure.

FSL_FLEXIO_I2S_DRIVER_VERSION

FlexIO I2S driver version 2.2.2.

FlexIO I2S transfer status.

Values:

enumerator kStatus_FLEXIO_I2S_Idle

FlexIO I2S is in idle state

enumerator kStatus_FLEXIO_I2S_TxBusy

FlexIO I2S Tx is busy

enumerator kStatus_FLEXIO_I2S_RxBusy

FlexIO I2S Tx is busy

enumerator kStatus_FLEXIO_I2S_Error

FlexIO I2S error occurred

enumerator kStatus_FLEXIO_I2S_QueueFull

FlexIO I2S transfer queue is full.

enumerator kStatus_FLEXIO_I2S_Timeout

FlexIO I2S timeout polling status flags.

enum _flexio_i2s_master_slave

Master or slave mode.

Values:

enumerator kFLEXIO_I2S_Master

Master mode

enumerator kFLEXIO_I2S_Slave

Slave mode

_flexio_i2s_interrupt_enable Define FlexIO FlexIO I2S interrupt mask.

Values:

enumerator kFLEXIO_I2S_TxDataRegEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_I2S_RxDataRegFullInterruptEnable
Receive buffer full interrupt enable.

_flexio_i2s_status_flags Define FlexIO FlexIO I2S status mask.

Values:

enumerator kFLEXIO_I2S_TxDataRegEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_I2S_RxDataRegFullFlag
Receive buffer full flag.

enum _flexio_i2s_sample_rate
Audio sample rate.

Values:

enumerator kFLEXIO_I2S_SampleRate8KHz
Sample rate 8000Hz

enumerator kFLEXIO_I2S_SampleRate11025Hz
Sample rate 11025Hz

enumerator kFLEXIO_I2S_SampleRate12KHz
Sample rate 12000Hz

enumerator kFLEXIO_I2S_SampleRate16KHz
Sample rate 16000Hz

enumerator kFLEXIO_I2S_SampleRate22050Hz
Sample rate 22050Hz

enumerator kFLEXIO_I2S_SampleRate24KHz
Sample rate 24000Hz

enumerator kFLEXIO_I2S_SampleRate32KHz
Sample rate 32000Hz

enumerator kFLEXIO_I2S_SampleRate44100Hz
Sample rate 44100Hz

enumerator kFLEXIO_I2S_SampleRate48KHz
Sample rate 48000Hz

enumerator kFLEXIO_I2S_SampleRate96KHz
Sample rate 96000Hz

enum _flexio_i2s_word_width
Audio word width.
Values:

enumerator kFLEXIO_I2S_WordWidth8bits
Audio data width 8 bits

enumerator kFLEXIO_I2S_WordWidth16bits
Audio data width 16 bits

enumerator kFLEXIO_I2S_WordWidth24bits
Audio data width 24 bits

enumerator kFLEXIO_I2S_WordWidth32bits
Audio data width 32 bits

typedef struct *_flexio_i2s_type* FLEXIO_I2S_Type
Define FlexIO I2S access structure typedef.

typedef enum *_flexio_i2s_master_slave* flexio_i2s_master_slave_t
Master or slave mode.

typedef struct *_flexio_i2s_config* flexio_i2s_config_t
FlexIO I2S configure structure.

typedef struct *_flexio_i2s_format* flexio_i2s_format_t
FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.

typedef enum *_flexio_i2s_sample_rate* flexio_i2s_sample_rate_t
Audio sample rate.

typedef enum *_flexio_i2s_word_width* flexio_i2s_word_width_t
Audio word width.

typedef struct *_flexio_i2s_transfer* flexio_i2s_transfer_t
Define FlexIO I2S transfer structure.

typedef struct *_flexio_i2s_handle* flexio_i2s_handle_t

typedef void (*flexio_i2s_callback_t)(*FLEXIO_I2S_Type* *base, *flexio_i2s_handle_t* *handle, *status_t* status, void *userData)
FlexIO I2S xfer callback prototype.

I2S_RETRY_TIMES
Retry times for waiting flag.

FLEXIO_I2S_XFER_QUEUE_SIZE

FlexIO I2S transfer queue size, user can refine it according to use case.

struct _flexio_i2s_type

#include <fsl_flexio_i2s.h> Define FlexIO I2S access structure typedef.

Public Members

FLEXIO_Type *flexioBase

FlexIO base pointer

uint8_t txPinIndex

Tx data pin index in FlexIO pins

uint8_t rxPinIndex

Rx data pin index

uint8_t bclkPinIndex

Bit clock pin index

uint8_t fsPinIndex

Frame sync pin index

uint8_t txShifterIndex

Tx data shifter index

uint8_t rxShifterIndex

Rx data shifter index

uint8_t bclkTimerIndex

Bit clock timer index

uint8_t fsTimerIndex

Frame sync timer index

struct _flexio_i2s_config

#include <fsl_flexio_i2s.h> FlexIO I2S configure structure.

Public Members

bool enableI2S

Enable FlexIO I2S

flexio_i2s_master_slave_t masterSlave

Master or slave

flexio_pin_polarity_t txPinPolarity
: Tx data pin polarity, active high or low

flexio_pin_polarity_t rxPinPolarity
: Rx data pin polarity

flexio_pin_polarity_t bclkPinPolarity
: Bit clock pin polarity

flexio_pin_polarity_t fsPinPolarity
: Frame sync pin polarity

flexio_shifter_timer_polarity_t txTimerPolarity
: Tx data valid on bclk rising or falling edge

flexio_shifter_timer_polarity_t rxTimerPolarity
: Rx data valid on bclk rising or falling edge

struct __flexio_i2s_format
: *#include <fsl_flexio_i2s.h>* FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx.

Public Members

uint8_t bitWidth
: Bit width of audio data, always 8/16/24/32 bits

uint32_t sampleRate_Hz
: Sample rate of the audio data

struct __flexio_i2s_transfer
: *#include <fsl_flexio_i2s.h>* Define FlexIO I2S transfer structure.

Public Members

uint8_t *data
: Data buffer start pointer

size_t dataSize
: Bytes to be transferred.

struct __flexio_i2s_handle
: *#include <fsl_flexio_i2s.h>* Define FlexIO I2S handle structure.

Public Members

uint32_t state
Internal state

flexio_i2s_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

uint8_t bitWidth
Bit width for transfer, 8/16/24/32bits

flexio_i2s_transfer_t queue[(4U)]
Transfer queue storing queued transfer

size_t transferSize[(4U)]
Data bytes need to transfer

volatile uint8_t queueUser
Index for user to queue transfer

volatile uint8_t queueDriver
Index for driver to get the transfer data and size

2.22 FlexIO MCU Interface LCD Driver

status_t FLEXIO_MCULCD_Init(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_config_t* *config, uint32_t srcClock_Hz)
Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO MCULCD hardware, and configures the FlexIO MCULCD with FlexIO MCULCD configuration. The configuration structure can be filled by the user, or be set with default values by the *FLEXIO_MCULCD_GetDefaultConfig*.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- config – Pointer to the flexio_mculcd_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Initialization success.
- kStatus_InvalidArgument – Initialization failed because of invalid argument.

void FLEXIO_MCULCD_Deinit(*FLEXIO_MCULCD_Type* *base)
Resets the FLEXIO_MCULCD timer and shifter configuration.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_GetDefaultConfig(*flexio_mculcd_config_t* *config)

Gets the default configuration to configure the FlexIO MCULCD.

The default configuration value is:

```
config->enable = true;
config->enableInDoze = false;
config->enableInDebug = true;
config->enableFastAccess = true;
config->baudRate_Bps = 96000000U;
```

Parameters

- config – Pointer to the flexio_mculcd_config_t structure.

uint32_t FLEXIO_MCULCD_GetStatusFlags(*FLEXIO_MCULCD_Type* *base)

Gets FlexIO MCULCD status flags.

Note: Don't use this function with DMA APIs.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.

Returns

status flag; OR'ed value or the *_flexio_mculcd_status_flags*.

void FLEXIO_MCULCD_ClearStatusFlags(*FLEXIO_MCULCD_Type* *base, uint32_t mask)

Clears FlexIO MCULCD status flags.

Note: Don't use this function with DMA APIs.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- mask – Status to clear, it is the OR'ed value of *_flexio_mculcd_status_flags*.

void FLEXIO_MCULCD_EnableInterrupts(*FLEXIO_MCULCD_Type* *base, uint32_t mask)

Enables the FlexIO MCULCD interrupt.

This function enables the FlexIO MCULCD interrupt.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- mask – Interrupts to enable, it is the OR'ed value of *_flexio_mculcd_interrupt_enable*.

void FLEXIO_MCULCD_DisableInterrupts(*FLEXIO_MCULCD_Type* *base, uint32_t mask)

Disables the FlexIO MCULCD interrupt.

This function disables the FlexIO MCULCD interrupt.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- mask – Interrupts to disable, it is the OR'ed value of *_flexio_mculcd_interrupt_enable*.

static inline void FLEXIO_MCULCD_EnableTxDMA(*FLEXIO_MCULCD_Type* *base, bool enable)

Enables/disables the FlexIO MCULCD transmit DMA.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- enable – True means enable DMA, false means disable DMA.

static inline void FLEXIO_MCULCD_EnableRxDMA(*FLEXIO_MCULCD_Type* *base, bool enable)

Enables/disables the FlexIO MCULCD receive DMA.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t FLEXIO_MCULCD_GetTxDataRegisterAddress(*FLEXIO_MCULCD_Type* *base)

Gets the FlexIO MCULCD transmit data register address.

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.

Returns

FlexIO MCULCD transmit data register address.

static inline uint32_t FLEXIO_MCULCD_GetRxDataRegisterAddress(*FLEXIO_MCULCD_Type* *base)

Gets the FlexIO MCULCD receive data register address.

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.

Returns

FlexIO MCULCD receive data register address.

status_t FLEXIO_MCULCD_SetBaudRate(*FLEXIO_MCULCD_Type* *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Set desired baud rate.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- baudRate_Bps – Desired baud rate in bit-per-second for all data lines combined.
- srcClock_Hz – FLEXIO clock frequency in Hz.

Return values

- kStatus_Success – Set successfully.
- kStatus_InvalidArgument – Could not set the baud rate.

void FLEXIO_MCULCD_SetSingleBeatWriteConfig(*FLEXIO_MCULCD_Type* *base)

Configures the FLEXIO MCULCD to multiple beats write mode.

At the begining multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by *FLEXIO_MCULCD_ClearSingleBeatWriteConfig*.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_ClearSingleBeatWriteConfig(*FLEXIO_MCULCD_Type* *base)

Clear the FLEXIO MCULCD multiple beats write mode configuration.

Clear the write configuration set by *FLEXIO_MCULCD_SetSingleBeatWriteConfig*.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_SetSingleBeatReadConfig(*FLEXIO_MCULCD_Type* *base)

Configures the FLEXIO MCULCD to multiple beats read mode.

At the begining or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by *FLEXIO_MCULCD_ClearSingleBeatReadConfig*.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_ClearSingleBeatReadConfig(*FLEXIO_MCULCD_Type* *base)

Clear the FLEXIO MCULCD multiple beats read mode configuration.

Clear the read configuration set by *FLEXIO_MCULCD_SetSingleBeatReadConfig*.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_SetMultiBeatsWriteConfig(*FLEXIO_MCULCD_Type* *base)

Configures the FLEXIO MCULCD to multiple beats write mode.

At the begining multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by FLEXIO_MCULCD_ClearMultBeatsWriteConfig.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_ClearMultiBeatsWriteConfig(*FLEXIO_MCULCD_Type* *base)

Clear the FLEXIO MCULCD multiple beats write mode configuration.

Clear the write configuration set by FLEXIO_MCULCD_SetMultBeatsWriteConfig.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_SetMultiBeatsReadConfig(*FLEXIO_MCULCD_Type* *base)

Configures the FLEXIO MCULCD to multiple beats read mode.

At the begining or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by FLEXIO_MCULCD_ClearMultBeatsReadConfig.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

void FLEXIO_MCULCD_ClearMultiBeatsReadConfig(*FLEXIO_MCULCD_Type* *base)

Clear the FLEXIO MCULCD multiple beats read mode configuration.

Clear the read configuration set by FLEXIO_MCULCD_SetMultBeatsReadConfig.

Note: This is an internal used function, upper layer should not use.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.

static inline void FLEXIO_MCULCD_Enable(*FLEXIO_MCULCD_Type* *base, bool enable)

Enables/disables the FlexIO MCULCD module operation.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type.
- enable – True to enable, false does not have any effect.

uint32_t FLEXIO_MCULCD_ReadData(*FLEXIO_MCULCD_Type* *base)

Read data from the FLEXIO MCULCD RX shifter buffer.

Read data from the RX shift buffer directly, it does no check whether the buffer is empty or not.

If the data bus width is 8-bit:

```
uint8_t value;
value = (uint8_t)FLEXIO_MCULCD_ReadData(base);
```

If the data bus width is 16-bit:

```
uint16_t value;
value = (uint16_t)FLEXIO_MCULCD_ReadData(base);
```

Note: This function returns the RX shifter buffer value (32-bit) directly. The return value should be converted according to data bus width.

Note: Don't use this function with DMA APIs.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.

Returns

The data read out.

static inline void FLEXIO_MCULCD_WriteData(*FLEXIO_MCULCD_Type* *base, uint32_t data)

Write data into the FLEXIO MCULCD TX shifter buffer.

Write data into the TX shift buffer directly, it does no check whether the buffer is full or not.

Note: Don't use this function with DMA APIs.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- data – The data to write.

static inline void FLEXIO_MCULCD_StartTransfer(*FLEXIO_MCULCD_Type* *base)

Assert the nCS to start transfer.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.

static inline void FLEXIO_MCULCD_StopTransfer(*FLEXIO_MCULCD_Type* *base)

De-assert the nCS to stop transfer.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.

void FLEXIO_MCULCD_WaitTransmitComplete(void)

Wait for transmit data send out finished.

Currently there is no effective method to wait for the data send out from the shiter, so here use a while loop to wait.

Note: This is an internal used function.

void FLEXIO_MCULCD_WriteCommandBlocking(*FLEXIO_MCULCD_Type* *base, uint32_t command)

Send command in blocking way.

This function sends the command and returns when the command has been sent out.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- command – The command to send.

void FLEXIO_MCULCD_WriteDataArrayBlocking(*FLEXIO_MCULCD_Type* *base, const void *data, size_t size)

Send data array in blocking way.

This function sends the data array and returns when the data sent out.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- data – The data array to send.
- size – How many bytes to write.

void FLEXIO_MCULCD_ReadDataArrayBlocking(*FLEXIO_MCULCD_Type* *base, void *data, size_t size)

Read data into array in blocking way.

This function reads the data into array and returns when the data read finished.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- data – The array to save the data.
- size – How many bytes to read.

void FLEXIO_MCULCD_WriteSameValueBlocking(*FLEXIO_MCULCD_Type* *base, uint32_t sameValue, size_t size)

Send the same value many times in blocking way.

This function sends the same value many times. It could be used to clear the LCD screen. If the data bus width is 8, this function will send LSB 8 bits of sameValue for size times. If the data bus is 16, this function will send LSB 16 bits of sameValue for size / 2 times.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- sameValue – The same value to send.
- size – How many bytes to send.

void FLEXIO_MCULCD_TransferBlocking(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_transfer_t* *xfer)

Performs a polling transfer.

Note: The API does not return until the transfer finished.

Parameters

- base – pointer to FLEXIO_MCULCD_Type structure.
- xfer – pointer to flexio_mculcd_transfer_t structure.

status_t FLEXIO_MCULCD_TransferCreateHandle(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_handle_t* *handle, *flexio_mculcd_transfer_callback_t* callback, void *userData)

Initializes the FlexIO MCULCD handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- handle – Pointer to the flexio_mculcd_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_MCULCD_TransferNonBlocking(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_handle_t* *handle, *flexio_mculcd_transfer_t* *xfer)

Transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- handle – Pointer to the flexio_mculcd_handle_t structure to store the transfer state.
- xfer – FlexIO MCULCD transfer structure. See *flexio_mculcd_transfer_t*.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_MCULCD_Busy – MCULCD is busy with another transfer.

void FLEXIO_MCULCD_TransferAbort(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_handle_t* *handle)

Aborts the data transfer, which used IRQ.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- handle – Pointer to the flexio_mculcd_handle_t structure to store the transfer state.

status_t FLEXIO_MCULCD_TransferGetCount(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_handle_t* *handle, size_t *count)

Gets the data transfer status which used IRQ.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- handle – Pointer to the flexio_mculcd_handle_t structure to store the transfer state.

- count – How many bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success – Get the transferred count Successfully.
- kStatus_NoTransferInProgress – No transfer in process.

void FLEXIO_MCULCD_TransferHandleIRQ(void *base, void *handle)

FlexIO MCULCD IRQ handler function.

Parameters

- base – Pointer to the FLEXIO_MCULCD_Type structure.
- handle – Pointer to the flexio_mculcd_handle_t structure to store the transfer state.

FSL_FLEXIO_MCULCD_DRIVER_VERSION

FlexIO MCULCD driver version.

FlexIO LCD transfer status.

Values:

enumerator kStatus_FLEXIO_MCULCD_Idle

FlexIO LCD is idle.

enumerator kStatus_FLEXIO_MCULCD_Busy

FlexIO LCD is busy

enumerator kStatus_FLEXIO_MCULCD_Error

FlexIO LCD error occurred

enum _flexio_mculcd_pixel_format

Define FlexIO MCULCD pixel format.

Values:

enumerator kFLEXIO_MCULCD_RGB565

RGB565, 16-bit.

enumerator kFLEXIO_MCULCD_BGR565

BGR565, 16-bit.

enumerator kFLEXIO_MCULCD_RGB888

RGB888, 24-bit.

enumerator kFLEXIO_MCULCD_BGR888

BGR888, 24-bit.

enum _flexio_mculcd_bus

Define FlexIO MCULCD bus type.

Values:

enumerator kFLEXIO_MCULCD_8080
Using Intel 8080 bus.

enumerator kFLEXIO_MCULCD_6800
Using Motorola 6800 bus.

enum _flexio_mculcd_interrupt_enable
Define FlexIO MCULCD interrupt mask.
Values:

enumerator kFLEXIO_MCULCD_TxEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_MCULCD_RxFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_mculcd_status_flags
Define FlexIO MCULCD status mask.
Values:

enumerator kFLEXIO_MCULCD_TxEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_MCULCD_RxFullFlag
Receive buffer full flag.

enum _flexio_mculcd_dma_enable
Define FlexIO MCULCD DMA mask.
Values:

enumerator kFLEXIO_MCULCD_TxDmaEnable
Tx DMA request source

enumerator kFLEXIO_MCULCD_RxDmaEnable
Rx DMA request source

enum _flexio_mculcd_transfer_mode
Transfer mode.
Values:

enumerator kFLEXIO_MCULCD_ReadArray
Read data into an array.

enumerator kFLEXIO_MCULCD_WriteArray
Write data from an array.

enumerator kFLEXIO_MCULCD_WriteSameValue
Write the same value many times.

typedef enum *_flexio_mculcd_pixel_format* flexio_mculcd_pixel_format_t

Define FlexIO MCULCD pixel format.

typedef enum *_flexio_mculcd_bus* flexio_mculcd_bus_t

Define FlexIO MCULCD bus type.

typedef void (*flexio_mculcd_pin_func_t)(bool set)

Function to set or clear the CS and RS pin.

typedef struct *_flexio_mculcd_type* FLEXIO_MCULCD_Type

Define FlexIO MCULCD access structure typedef.

typedef struct *_flexio_mculcd_config* flexio_mculcd_config_t

Define FlexIO MCULCD configuration structure.

typedef enum *_flexio_mculcd_transfer_mode* flexio_mculcd_transfer_mode_t

Transfer mode.

typedef struct *_flexio_mculcd_transfer* flexio_mculcd_transfer_t

Define FlexIO MCULCD transfer structure.

typedef struct *_flexio_mculcd_handle* flexio_mculcd_handle_t

typedef for flexio_mculcd_handle_t in advance.

typedef void (*flexio_mculcd_transfer_callback_t)(*FLEXIO_MCULCD_Type* *base, *flexio_mculcd_handle_t* *handle, *status_t* status, void *userData)

FlexIO MCULCD callback for finished transfer.

When transfer finished, the callback function is called and returns the status as kStatus_FLEXIO_MCULCD_Idle.

FLEXIO_MCULCD_WAIT_COMPLETE_TIME

The delay time to wait for FLEXIO transmit complete.

Currently there is no method to detect whether the data has been sent out from the shifter, so the driver use a software delay for this. When the data is written to shifter buffer, the driver call the delay function to wait for the data shift out. If this value is too small, then the last few bytes might be lost when writing data using interrupt method or DMA method.

FLEXIO_MCULCD_DATA_BUS_WIDTH

The data bus width, must be 8 or 16.

FLEXIO_MCULCD_LEGACY_GPIO_FUNC

Whether to use legacy GPIO functions to control the CS/RS/RDWR pin signal.

If using the legacy pin functions, there is no user defined argument passed to the function.

struct _flexio_mculcd_type

#include <fsl_flexio_mculcd.h> Define FlexIO MCULCD access structure typedef.

Public Members

FLEXIO_Type *flexioBase

FlexIO base pointer.

flexio_mculcd_bus_t busType

The bus type, 8080 or 6800.

uint8_t dataPinStartIndex

Start index of the data pin, the FlexIO pin dataPinStartIndex to (dataPinStartIndex + FLEXIO_MCULCD_DATA_BUS_WIDTH -1) will be used for data transfer. Only support data bus width 8 and 16.

uint8_t ENWRPinIndex

Pin select for WR(8080 mode), EN(6800 mode).

uint8_t RDPinIndex

Pin select for RD(8080 mode), not used in 6800 mode.

uint8_t txShifterStartIndex

Start index of shifters used for data write, it must be 0 or 4.

uint8_t txShifterEndIndex

End index of shifters used for data write.

uint8_t rxShifterStartIndex

Start index of shifters used for data read.

uint8_t rxShifterEndIndex

End index of shifters used for data read, it must be 3 or 7.

uint8_t timerIndex

Timer index used in FlexIO MCULCD.

flexio_mculcd_pin_func_t setCSPin

Function to set or clear the CS pin.

flexio_mculcd_pin_func_t setRSPin

Function to set or clear the RS pin.

flexio_mculcd_pin_func_t setRDWRPin

Function to set or clear the RD/WR pin, only used in 6800 mode.

struct __flexio_mculcd_config

#include <fsl_flexio_mculcd.h> Define FlexIO MCULCD configuration structure.

Public Members

bool enable

Enable/disable FlexIO MCULCD after configuration.

bool enableInDoze

Enable/disable FlexIO operation in doze mode.

bool enableInDebug

Enable/disable FlexIO operation in debug mode.

bool enableFastAccess

Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps

Baud rate in bit-per-second for all data lines combined.

struct _flexio_mculcd_transfer

#include <fsl_flexio_mculcd.h> Define FlexIO MCULCD transfer structure.

Public Members

uint32_t command

Command to send.

uint32_t dataAddrOrSameValue

When sending the same value for many times, this is the value to send. When writing or reading array, this is the address of the data array.

size_t dataSize

How many bytes to transfer.

flexio_mculcd_transfer_mode_t mode

Transfer mode.

bool dataOnly

Send data only when tx without the command.

struct _flexio_mculcd_handle

#include <fsl_flexio_mculcd.h> Define FlexIO MCULCD handle structure.

Public Members

uint32_t dataAddrOrSameValue

When sending the same value for many times, this is the value to send. When writing or reading array, this is the address of the data array.

size_t dataCount
Total count to be transferred.

volatile size_t remainingCount
Remaining count to transfer.

volatile uint32_t state
FlexIO MCULCD internal state.

flexio_mculcd_transfer_callback_t completionCallback
FlexIO MCULCD transfer completed callback.

void *userData
Callback parameter.

2.23 FlexIO SPI Driver

void FLEXIO_SPI_MasterInit(*FLEXIO_SPI_Type* *base, *flexio_spi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration. The configuration structure can be filled by the user, or be set with default values by the *FLEXIO_SPI_MasterGetDefaultConfig()*.

Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
.enableMaster = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 500000,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Note: 1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 2*2=4. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- masterConfig – Pointer to the flexio_spi_master_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

void FLEXIO_SPI_MasterDeinit(*FLEXIO_SPI_Type* *base)

Resets the FlexIO SPI timer and shifter config.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_MasterGetDefaultConfig(*flexio_spi_master_config_t* *masterConfig)

Gets the default configuration to configure the FlexIO SPI master. The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – Pointer to the flexio_spi_master_config_t structure.

void FLEXIO_SPI_SlaveInit(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_config_t* *slaveConfig)

Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration. The configuration structure can be filled by the user, or be set with default values by the *FLEXIO_SPI_SlaveGetDefaultConfig()*.

Note: 1.Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2.FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 3*2=6. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8. Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0}
};
flexio_spi_slave_config_t config = {
.enableSlave = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

void FLEXIO_SPI_SlaveDeinit(*FLEXIO_SPI_Type* *base)

Gates the FlexIO clock.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.

void FLEXIO_SPI_SlaveGetDefaultConfig(*flexio_spi_slave_config_t* *slaveConfig)

Gets the default configuration to configure the FlexIO SPI slave. The configuration can be used directly for calling the FLEXIO_SPI_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – Pointer to the flexio_spi_slave_config_t structure.

uint32_t FLEXIO_SPI_GetStatusFlags(*FLEXIO_SPI_Type* *base)

Gets FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_ClearStatusFlags(*FLEXIO_SPI_Type* *base, uint32_t mask)

Clears FlexIO SPI status flags.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – status flag The parameter can be any combination of the following values:
  - kFLEXIO_SPI_TxEmptyFlag
  - kFLEXIO_SPI_RxEmptyFlag

void FLEXIO_SPI_EnableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Enables the FlexIO SPI interrupt.

This function enables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source. The parameter can be any combination of the following values:
  - kFLEXIO_SPI_RxFullInterruptEnable

- kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_DisableInterrupts(*FLEXIO_SPI_Type* *base, uint32_t mask)

Disables the FlexIO SPI interrupt.

This function disables the FlexIO SPI interrupt.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – interrupt source The parameter can be any combination of the following values:
  - kFLEXIO_SPI_RxFullInterruptEnable
  - kFLEXIO_SPI_TxEmptyInterruptEnable

void FLEXIO_SPI_EnableDMA(*FLEXIO_SPI_Type* *base, uint32_t mask, bool enable)

Enables/disables the FlexIO SPI transmit DMA. This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO_SPI_TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- mask – SPI DMA source.
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t FLEXIO_SPI_GetTxDataRegisterAddress(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Gets the FlexIO SPI transmit data register address for MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI transmit data register address.

static inline uint32_t FLEXIO_SPI_GetRxDataRegisterAddress(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Gets the FlexIO SPI receive data register address for the MSB first transfer.

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

FlexIO SPI receive data register address.

static inline void FLEXIO_SPI_Enable(*FLEXIO_SPI_Type* *base, bool enable)

Enables/disables the FlexIO SPI module operation.

Parameters

- base – Pointer to the FLEXIO_SPI_Type.
- enable – True to enable, false does not have any effect.

void FLEXIO_SPI_MasterSetBaudRate(*FLEXIO_SPI_Type* *base, uint32_t baudRate_Bps, uint32_t srcClockHz)

Sets baud rate for the FlexIO SPI transfer, which is only used for the master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- baudRate_Bps – Baud Rate needed in Hz.
- srcClockHz – SPI source clock frequency in Hz.

static inline void FLEXIO_SPI_WriteData(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, uint32_t data)

Writes one byte of data, which is sent using the MSB method.

Note: This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- data – 8/16/32 bit data.

static inline uint32_t FLEXIO_SPI_ReadData(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction)

Reads 8 bit/16 bit data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.

Returns

8 bit/16 bit data received.

status_t FLEXIO_SPI_WriteBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, const uint8_t *buffer, size_t size)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_ReadBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_shift_direction_t* direction, uint8_t *buffer, size_t size)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- direction – Shift direction of MSB first or LSB first.
- buffer – The buffer to store the received bytes.
- size – The number of data bytes to be received.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

status_t FLEXIO_SPI_MasterTransferBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_transfer_t* *xfer)

Receives a buffer of bytes.

Note: This function blocks via polling until all bytes have been received.

Parameters

- base – pointer to FLEXIO_SPI_Type structure
- xfer – FlexIO SPI transfer structure, see *flexio_spi_transfer_t*.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_FLEXIO_SPI_Timeout – The transfer timed out and was aborted.

void FLEXIO_SPI_FlushShifters(*FLEXIO_SPI_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

status_t FLEXIO_SPI_MasterTransferCreateHandle(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_master_transfer_callback_t* callback, void *userData)

Initializes the FlexIO SPI Master handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.

- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_SPI_MasterTransferNonBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Master transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- xfer – FlexIO SPI transfer structure. See *flexio_spi_transfer_t*.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle, is running another transfer.

void FLEXIO_SPI_MasterTransferAbort(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle)

Aborts the master data transfer, which used IRQ.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

status_t FLEXIO_SPI_MasterTransferGetCount(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, size_t *count)

Gets the data transfer status which used IRQ.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXIO_SPI_MasterTransferHandleIRQ(void *spiType, void *spiHandle)

FlexIO SPI master IRQ handler function.

Parameters

- spiType – Pointer to the FLEXIO_SPI_Type structure.
- spiHandle – Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

status_t FLEXIO_SPI_SlaveTransferCreateHandle(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, *flexio_spi_slave_transfer_callback_t* callback, void *userData)

Initializes the FlexIO SPI Slave handle, which is used in transactional functions.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

status_t FLEXIO_SPI_SlaveTransferNonBlocking(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, *flexio_spi_transfer_t* *xfer)

Slave transfer data using IRQ.

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- base – Pointer to the FLEXIO_SPI_Type structure.
- xfer – FlexIO SPI transfer structure. See *flexio_spi_transfer_t*.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_FLEXIO_SPI_Busy – SPI is not idle; it is running another transfer.

static inline void FLEXIO_SPI_SlaveTransferAbort(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle)

Aborts the slave data transfer which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

static inline *status_t* FLEXIO_SPI_SlaveTransferGetCount(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, size_t *count)

Gets the data transfer status which used IRQ, share same API with master.

Parameters

- base – Pointer to the FLEXIO_SPI_Type structure.
- handle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void FLEXIO_SPI_SlaveTransferHandleIRQ(void *spiType, void *spiHandle)

FlexIO SPI slave IRQ handler function.

Parameters

- spiType – Pointer to the FLEXIO_SPI_Type structure.
- spiHandle – Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

FSL_FLEXIO_SPI_DRIVER_VERSION

FlexIO SPI driver version.

Error codes for the FlexIO SPI driver.

Values:

enumerator kStatus_FLEXIO_SPI_Busy

FlexIO SPI is busy.

enumerator kStatus_FLEXIO_SPI_Idle

SPI is idle

enumerator kStatus_FLEXIO_SPI_Error

FlexIO SPI error.

enumerator kStatus_FLEXIO_SPI_Timeout

FlexIO SPI timeout polling status flags.

enum _flexio_spi_clock_phase

FlexIO SPI clock phase configuration.

Values:

enumerator kFLEXIO_SPI_ClockPhaseFirstEdge

First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

enumerator kFLEXIO_SPI_ClockPhaseSecondEdge
First edge on SPSCK occurs at the start of the first cycle of a data transfer.

enum _flexio_spi_shift_direction
FlexIO SPI data shifter direction options.
Values:

enumerator kFLEXIO_SPI_MsbFirst
Data transfers start with most significant bit.

enumerator kFLEXIO_SPI_LsbFirst
Data transfers start with least significant bit.

enum _flexio_spi_data_bitcount_mode
FlexIO SPI data length mode options.
Values:

enumerator kFLEXIO_SPI_8BitMode
8-bit data transmission mode.

enumerator kFLEXIO_SPI_16BitMode
16-bit data transmission mode.

enumerator kFLEXIO_SPI_32BitMode
32-bit data transmission mode.

enum _flexio_spi_interrupt_enable
Define FlexIO SPI interrupt mask.
Values:

enumerator kFLEXIO_SPI_TxEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_SPI_RxFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_spi_status_flags
Define FlexIO SPI status mask.
Values:

enumerator kFLEXIO_SPI_TxBufferEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_SPI_RxBufferFullFlag
Receive buffer full flag.

enum _flexio_spi_dma_enable

Define FlexIO SPI DMA mask.

Values:

enumerator kFLEXIO_SPI_TxDmaEnable

Tx DMA request source

enumerator kFLEXIO_SPI_RxDmaEnable

Rx DMA request source

enumerator kFLEXIO_SPI_DmaAllEnable

All DMA request source

enum _flexio_spi_transfer_flags

Define FlexIO SPI transfer flags.

Note: Use kFLEXIO_SPI_csContinuous and one of the other flags to OR together to form the transfer flag.

Values:

enumerator kFLEXIO_SPI_8bitMsb

FlexIO SPI 8-bit MSB first

enumerator kFLEXIO_SPI_8bitLsb

FlexIO SPI 8-bit LSB first

enumerator kFLEXIO_SPI_16bitMsb

FlexIO SPI 16-bit MSB first

enumerator kFLEXIO_SPI_16bitLsb

FlexIO SPI 16-bit LSB first

enumerator kFLEXIO_SPI_32bitMsb

FlexIO SPI 32-bit MSB first

enumerator kFLEXIO_SPI_32bitLsb

FlexIO SPI 32-bit LSB first

enumerator kFLEXIO_SPI_csContinuous

Enable the CS signal continuous mode

typedef enum *_flexio_spi_clock_phase* flexio_spi_clock_phase_t

FlexIO SPI clock phase configuration.

typedef enum *_flexio_spi_shift_direction* flexio_spi_shift_direction_t

FlexIO SPI data shifter direction options.

typedef enum *_flexio_spi_data_bitcount_mode* flexio_spi_data_bitcount_mode_t

FlexIO SPI data length mode options.

typedef struct *_flexio_spi_type* FLEXIO_SPI_Type

Define FlexIO SPI access structure typedef.

typedef struct *_flexio_spi_master_config* flexio_spi_master_config_t

Define FlexIO SPI master configuration structure.

typedef struct *_flexio_spi_slave_config* flexio_spi_slave_config_t

Define FlexIO SPI slave configuration structure.

typedef struct *_flexio_spi_transfer* flexio_spi_transfer_t

Define FlexIO SPI transfer structure.

typedef struct *_flexio_spi_master_handle* flexio_spi_master_handle_t

typedef for flexio_spi_master_handle_t in advance.

typedef *flexio_spi_master_handle_t* flexio_spi_slave_handle_t

Slave handle is the same with master handle.

typedef void (*flexio_spi_master_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_master_handle_t* *handle, *status_t* status, void *userData)

FlexIO SPI master callback for finished transmit.

typedef void (*flexio_spi_slave_transfer_callback_t)(*FLEXIO_SPI_Type* *base, *flexio_spi_slave_handle_t* *handle, *status_t* status, void *userData)

FlexIO SPI slave callback for finished transmit.

FLEXIO_SPI_DUMMYDATA

FlexIO SPI dummy transfer data, the data is sent while txData is NULL.

SPI_RETRY_TIMES

Retry times for waiting flag.

FLEXIO_SPI_XFER_DATA_FORMAT(flag)

Get the transfer data format of width and bit order.

struct _flexio_spi_type

#include <fsl_flexio_spi.h> Define FlexIO SPI access structure typedef.

Public Members

FLEXIO_Type *flexioBase

FlexIO base pointer.

uint8_t SDOPinIndex

Pin select for data output. To set SDO pin in Hi-Z state, user needs to mux the pin as GPIO input and disable all pull up/down in application.

uint8_t SDIPinIndex
: Pin select for data input.

uint8_t SCKPinIndex
: Pin select for clock.

uint8_t CSnPinIndex
: Pin select for enable.

uint8_t shifterIndex[2]
: Shifter index used in FlexIO SPI.

uint8_t timerIndex[2]
: Timer index used in FlexIO SPI.

struct __flexio_spi_master_config
: *#include <fsl_flexio_spi.h>* Define FlexIO SPI master configuration structure.

Public Members

bool enableMaster
: Enable/disable FlexIO SPI master after configuration.

bool enableInDoze
: Enable/disable FlexIO operation in doze mode.

bool enableInDebug
: Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
: Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
: Baud rate in Bps.

flexio_spi_clock_phase_t phase
: Clock phase.

flexio_spi_data_bitcount_mode_t dataMode
: 8bit or 16bit mode.

struct __flexio_spi_slave_config
: *#include <fsl_flexio_spi.h>* Define FlexIO SPI slave configuration structure.

Public Members

bool enableSlave
Enable/disable FlexIO SPI slave after configuration.

bool enableInDoze
Enable/disable FlexIO operation in doze mode.

bool enableInDebug
Enable/disable FlexIO operation in debug mode.

bool enableFastAccess
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

flexio_spi_clock_phase_t phase
Clock phase.

flexio_spi_data_bitcount_mode_t dataMode
8bit or 16bit mode.

struct __flexio_spi_transfer
#include <fsl_flexio_spi.h> Define FlexIO SPI transfer structure.

Public Members

const uint8_t *txData
Send buffer.

uint8_t *rxData
Receive buffer.

size_t dataSize
Transfer bytes.

uint8_t flags
FlexIO SPI control flag, MSB first or LSB first.

struct __flexio_spi_master_handle
#include <fsl_flexio_spi.h> Define FlexIO SPI handle structure.

Public Members

const uint8_t *txData
Transfer buffer.

uint8_t *rxData
Receive buffer.

size_t transferSize
Total bytes to be transferred.

volatile size_t txRemainingBytes
Send data remaining in bytes.

volatile size_t rxRemainingBytes
Receive data remaining in bytes.

volatile uint32_t state
FlexIO SPI internal state.

uint8_t bytePerFrame
SPI mode, 2bytes or 1byte in a frame

flexio_spi_shift_direction_t direction
Shift direction.

flexio_spi_master_transfer_callback_t callback
FlexIO SPI callback.

void *userData
Callback parameter.

bool isCsContinuous
Is current transfer using CS continuous mode.

uint32_t timer1Cfg
TIMER1 TIMCFG regiser value backup.

2.24 FlexIO UART Driver

status_t FLEXIO_UART_Init(*FLEXIO_UART_Type* *base, const *flexio_uart_config_t* *userConfig, uint32_t srcClock_Hz)

Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration. The configuration structure can be filled by the user or be set with default values by *FLEXIO_UART_GetDefaultConfig()*.

Example

```
FLEXIO_UART_Type base = {
.flexioBase = FLEXIO,
.TxPinIndex = 0,
.RxPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
```

(continues on next page)

(continued from previous page)

```
flexio_uart_config_t config = {
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 115200U,
.bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- userConfig – Pointer to the flexio_uart_config_t structure.
- srcClock_Hz – FlexIO source clock in Hz.

Return values

- kStatus_Success – Configuration success.
- kStatus_FLEXIO_UART_BaudrateNotSupport – Baudrate is not supported for current clock source frequency.

void FLEXIO_UART_Deinit(*FLEXIO_UART_Type* *base)

Resets the FlexIO UART shifter and timer config.

Note: After calling this API, call the FLEXO_UART_Init to use the FlexIO UART module.

Parameters

- base – Pointer to FLEXIO_UART_Type structure

void FLEXIO_UART_GetDefaultConfig(*flexio_uart_config_t* *userConfig)

Gets the default configuration to configure the FlexIO UART. The configuration can be used directly for calling the *FLEXIO_UART_Init()*. Example:

```
flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

- userConfig – Pointer to the flexio_uart_config_t structure.

uint32_t FLEXIO_UART_GetStatusFlags(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART status flags.

void FLEXIO_UART_ClearStatusFlags(*FLEXIO_UART_Type* *base, uint32_t mask)

Gets the FlexIO UART status flags.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

- mask – Status flag. The parameter can be any combination of the following values:
  - kFLEXIO_UART_TxDataRegEmptyFlag
  - kFLEXIO_UART_RxEmptyFlag
  - kFLEXIO_UART_RxOverRunFlag

void FLEXIO_UART_EnableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Enables the FlexIO UART interrupt.

This function enables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

void FLEXIO_UART_DisableInterrupts(*FLEXIO_UART_Type* *base, uint32_t mask)

Disables the FlexIO UART interrupt.

This function disables the FlexIO UART interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- mask – Interrupt source.

static inline uint32_t FLEXIO_UART_GetTxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UARt transmit data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART transmit data register address.

static inline uint32_t FLEXIO_UART_GetRxDataRegisterAddress(*FLEXIO_UART_Type* *base)

Gets the FlexIO UART receive data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART receive data register address.

static inline void FLEXIO_UART_EnableTxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART transmit DMA. This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO_UART_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_EnableRxDMA(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART receive DMA. This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO_UART_RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- enable – True to enable, false to disable.

static inline void FLEXIO_UART_Enable(*FLEXIO_UART_Type* *base, bool enable)

Enables/disables the FlexIO UART module operation.

Parameters

- base – Pointer to the FLEXIO_UART_Type.
- enable – True to enable, false does not have any effect.

static inline void FLEXIO_UART_WriteByte(*FLEXIO_UART_Type* *base, const uint8_t *buffer)

Writes one byte of data.

Note: This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The data bytes to send.

static inline void FLEXIO_UART_ReadByte(*FLEXIO_UART_Type* *base, uint8_t *buffer)

Reads one byte of data.

Note: This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- buffer – The buffer to store the received bytes.

status_t FLEXIO_UART_WriteBlocking(*FLEXIO_UART_Type* *base, const uint8_t *txData, size_t txSize)

Sends a buffer of data bytes.

Note: This function blocks using the polling method until all bytes have been sent.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- txData – The data bytes to send.
- txSize – The number of data bytes to send.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t FLEXIO_UART_ReadBlocking(*FLEXIO_UART_Type* *base, uint8_t *rxData, size_t rxSize)

Receives a buffer of bytes.

Note: This function blocks using the polling method until all bytes have been received.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- rxData – The buffer to store the received bytes.
- rxSize – The number of data bytes to be received.

Return values

- kStatus_FLEXIO_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t FLEXIO_UART_TransferCreateHandle(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_callback_t* callback, void *userData)

Initializes the UART handle.

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the "background" receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the *FLEXIO_UART_TransferReceiveNonBlocking()* API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – to FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- callback – The callback function.
- userData – The parameter of the callback function.

Return values

- kStatus_Success – Successfully create the handle.
- kStatus_OutOfRange – The FlexIO type/handle/ISR table out of range.

void FLEXIO_UART_TransferStartRingBuffer(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_ReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note: When using the RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, only 31 bytes are used for saving data.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – Size of the ring buffer.

void FLEXIO_UART_TransferStopRingBuffer(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferSendNonBlocking(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the *kStatus_FLEXIO_UART_TxIdle* as status parameter.

Note: The kStatus_FLEXIO_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – FlexIO UART transfer structure. See *flexio_uart_transfer_t*.

Return values

- kStatus_Success – Successfully starts the data transmission.
- kStatus_UART_TxBusy – Previous transmission still not finished, data not written to the TX register.

void FLEXIO_UART_TransferAbortSend(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. Get the remainBytes to find out how many bytes are still not sent out.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferGetSendCount(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, size_t *count)

Gets the number of bytes sent.

This function gets the number of bytes sent driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes sent so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

status_t FLEXIO_UART_TransferReceiveNonBlocking(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *flexio_uart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter kStatus_UART_RxIdle. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to xfer->data. This function returns with the parameter receivedBytes set to 5. For the last 5 bytes, newly arrived data is saved from the xfer->data[5]. When 5 bytes are received, the UART driver notifies upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- xfer – UART transfer structure. See *flexio_uart_transfer_t*.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.

- kStatus_FLEXIO_UART_RxBusy – Previous receive request is not finished.

void FLEXIO_UART_TransferAbortReceive(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle)

Aborts the receive data which was using IRQ.

This function aborts the receive data which was using IRQ.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

status_t FLEXIO_UART_TransferGetReceiveCount(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, size_t *count)

Gets the number of bytes received.

This function gets the number of bytes received driven by interrupt.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.
- handle – Pointer to the flexio_uart_handle_t structure to store the transfer state.
- count – Number of bytes received so far by the non-blocking transaction.

Return values

- kStatus_NoTransferInProgress – transfer has finished or no transfer in progress.
- kStatus_Success – Successfully return the count.

void FLEXIO_UART_TransferHandleIRQ(void *uartType, void *uartHandle)

FlexIO UART IRQ handler function.

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

- uartType – Pointer to the FLEXIO_UART_Type structure.
- uartHandle – Pointer to the flexio_uart_handle_t structure to store the transfer state.

void FLEXIO_UART_FlushShifters(*FLEXIO_UART_Type* *base)

Flush tx/rx shifters.

Parameters

- base – Pointer to the FLEXIO_UART_Type structure.

FSL_FLEXIO_UART_DRIVER_VERSION

FlexIO UART driver version.

Error codes for the UART driver.

Values:

enumerator kStatus_FLEXIO_UART_TxBusy
Transmitter is busy.

enumerator kStatus_FLEXIO_UART_RxBusy
Receiver is busy.

enumerator kStatus_FLEXIO_UART_TxIdle
UART transmitter is idle.

enumerator kStatus_FLEXIO_UART_RxIdle
UART receiver is idle.

enumerator kStatus_FLEXIO_UART_ERROR
ERROR happens on UART.

enumerator kStatus_FLEXIO_UART_RxRingBufferOverrun
UART RX software ring buffer overrun.

enumerator kStatus_FLEXIO_UART_RxHardwareOverrun
UART RX receiver overrun.

enumerator kStatus_FLEXIO_UART_Timeout
UART times out.

enumerator kStatus_FLEXIO_UART_BaudrateNotSupport
Baudrate is not supported in current clock source

enum _flexio_uart_bit_count_per_char
FlexIO UART bit count per char.
Values:

enumerator kFLEXIO_UART_7BitsPerChar
7-bit data characters

enumerator kFLEXIO_UART_8BitsPerChar
8-bit data characters

enumerator kFLEXIO_UART_9BitsPerChar
9-bit data characters

enum _flexio_uart_interrupt_enable
Define FlexIO UART interrupt mask.
Values:

enumerator kFLEXIO_UART_TxDataRegEmptyInterruptEnable
Transmit buffer empty interrupt enable.

enumerator kFLEXIO_UART_RxDataRegFullInterruptEnable
Receive buffer full interrupt enable.

enum _flexio_uart_status_flags
Define FlexIO UART status mask.
Values:

enumerator kFLEXIO_UART_TxDataRegEmptyFlag
Transmit buffer empty flag.

enumerator kFLEXIO_UART_RxDataRegFullFlag
Receive buffer full flag.

enumerator kFLEXIO_UART_RxOverRunFlag
Receive buffer over run flag.

typedef enum *_flexio_uart_bit_count_per_char* flexio_uart_bit_count_per_char_t
FlexIO UART bit count per char.

typedef struct *_flexio_uart_type* FLEXIO_UART_Type
Define FlexIO UART access structure typedef.

typedef struct *_flexio_uart_config* flexio_uart_config_t
Define FlexIO UART user configuration structure.

typedef struct *_flexio_uart_transfer* flexio_uart_transfer_t
Define FlexIO UART transfer structure.

typedef struct *_flexio_uart_handle* flexio_uart_handle_t

typedef void (*flexio_uart_transfer_callback_t)(*FLEXIO_UART_Type* *base, *flexio_uart_handle_t* *handle, *status_t* status, void *userData)
FlexIO UART transfer callback function.

UART_RETRY_TIMES
Retry times for waiting flag.

struct _flexio_uart_type
#include <fsl_flexio_uart.h> Define FlexIO UART access structure typedef.

Public Members

FLEXIO_Type *flexioBase
FlexIO base pointer.

uint8_t TxPinIndex
Pin select for UART_Tx.

uint8_t RxPinIndex
: Pin select for UART_Rx.

uint8_t shifterIndex[2]
: Shifter index used in FlexIO UART.

uint8_t timerIndex[2]
: Timer index used in FlexIO UART.

struct _flexio_uart_config
: *#include <fsl_flexio_uart.h>* Define FlexIO UART user configuration structure.

Public Members

bool enableUart
: Enable/disable FlexIO UART TX & RX.

bool enableInDoze
: Enable/disable FlexIO operation in doze mode

bool enableInDebug
: Enable/disable FlexIO operation in debug mode

bool enableFastAccess
: Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

uint32_t baudRate_Bps
: Baud rate in Bps.

flexio_uart_bit_count_per_char_t bitCountPerChar
: number of bits, 7/8/9 -bit

struct _flexio_uart_transfer
: *#include <fsl_flexio_uart.h>* Define FlexIO UART transfer structure.

Public Members

size_t dataSize
: Transfer size

struct _flexio_uart_handle
: *#include <fsl_flexio_uart.h>* Define FLEXIO UART handle structure.

Public Members

const uint8_t *volatile txData
: Address of remaining data to send.

volatile size_t txDataSize
: Size of the remaining data to send.

uint8_t *volatile rxData
: Address of remaining data to receive.

volatile size_t rxDataSize
: Size of the remaining data to receive.

size_t txDataSizeAll
: Total bytes to be sent.

size_t rxDataSizeAll
: Total bytes to be received.

uint8_t *rxRingBuffer
: Start address of the receiver ring buffer.

size_t rxRingBufferSize
: Size of the ring buffer.

volatile uint16_t rxRingBufferHead
: Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
: Index for the user to get data from the ring buffer.

flexio_uart_transfer_callback_t callback
: Callback function.

void *userData
: UART callback function parameter.

volatile uint8_t txState
: TX transfer state.

volatile uint8_t rxState
: RX transfer state

union ___unnamed73___

Public Members

uint8_t *data
The buffer of data to be transfer.

uint8_t *rxData
The buffer to receive data.

const uint8_t *txData
The buffer of data to be sent.

2.25 FREQME: Frequency Measurement

2.26 GDET

status_t GDET_Init(GDET_Type *base)
Initialize GDET.

This function initializes GDET block and setting.

Parameters

- base – GDET peripheral base address

Returns
Status of the init operation

void GDET_Deinit(GDET_Type *base)
Deinitialize GDET.

This function deinitializes GDET secure counter.

Parameters

- base – GDET peripheral base address

status_t GDET_Enable(GDET_Type *base)
Enable GDET.

This function enables GDET and interrupts.

Parameters

- base – GDET peripheral base address

Returns
Status of the enable operation

status_t GDET_Disable(GDET_Type *base)
Disable GDET.

This function disables GDET and interrupts.

Parameters

- base – GDET peripheral base address

Returns
Status of the disable operation

status_t GDET_IsolateOn(GDET_Type *base)

Turn on GDET isolation.

This function turns on isolation of GDET peripheral

Parameters

- base – GDET peripheral base address

Returns

Status of the operation

status_t GDET_IsolateOff(GDET_Type *base)

Turn off GDET isolation.

This function turns off isolation of GDET peripheral

Parameters

- base – GDET peripheral base address

Returns

Status of the operation

status_t GDET_ReconfigureVoltageMode(GDET_Type *base, *gdet_core_voltage_t* voltage)

Change expected core voltage.

This function changes core voltage which Glitch detector expect.

Parameters

- base – GDET peripheral base address
- voltage – Expected core voltage

Returns

Status of the GDET reconfiguration operation

FSL_GDET_DRIVER_VERSION

Defines GDET driver version 2.1.4.

Change log:

- Version 2.1.4
  - Add wait into GDET_ReconfigureVoltageMode to allow update conf properly finish
- Version 2.1.3
  - Update to KW43 GDET offering
- Version 2.1.2
  - Update to MCXA577 GDET offering
- Version 2.1.1
  - Update to RT700 GDET offering
- Version 2.1.0
  - Update for multiple instances
  - Fix bug in isolation off API
  - Add enable and disable APIs
- 2.0.1
  - Fix MISRA in *GDET_ReconfigureVoltageMode()*

- Version 2.0.0
  - initial version

typedef uint32_t gdet_core_voltage_t

GDET Core Voltage.

These constants are used to define core voltage argument to be used with *GDET_ReconfigureVoltageMode()*. Different SoC may support various volatages, refer to documentation.

void GDET_DriverIRQHandler(void)

kGDET_0_9v

Voltage (0.9V)

kGDET_1_0v

Voltage (1.0V)

kGDET_1_1v

Voltage (1.1V)

2.27 Glitch Detect

2.28 GLIKEY

2.29 GLIKEY

Values:

enumerator kStatus_GLIKEY_LockedError

GLIKEY status for locked SFR registers (unexpected) .

enumerator kStatus_GLIKEY_NotLocked

GLIKEY status for unlocked SFR registers.

enumerator kStatus_GLIKEY_Locked

GLIKEY status for locked SFR registers.

enumerator kStatus_GLIKEY_DisabledError

GLIKEY status for disabled error.

FSL_GLIKEY_DRIVER_VERSION

Defines GLIKEY driver version 2.0.1.

Change log:

- Version 2.0.1

- Implement INIT state recovery from the LOCKED state after a reset when the previous index was locked.

- Version 2.0.0
  - Initial version

GLIKEY_CODEWORD_STEP1

GLIKEY_CODEWORD_STEP2

GLIKEY_CODEWORD_STEP3

GLIKEY_CODEWORD_STEP4

GLIKEY_CODEWORD_STEP5

GLIKEY_CODEWORD_STEP6

GLIKEY_CODEWORD_STEP7

GLIKEY_CODEWORD_STEP_EN

GLIKEY_FSM_WR_DIS

GLIKEY_FSM_INIT

GLIKEY_FSM_STEP1

GLIKEY_FSM_STEP2

GLIKEY_FSM_STEP3

GLIKEY_FSM_STEP4

GLIKEY_FSM_LOCKED

GLIKEY_FSM_WR_EN

GLIKEY_FSM_SSR_RESET

uint32_t GLIKEY_GetStatus(GLIKEY_Type *base)

Retreives the current status of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Glikey status information

status_t GLIKEY_IsLocked(GLIKEY_Type *base)

Get if Glikey is locked.

This operation returns the locking status of Glikey.

Return values

- kStatus_GLIKEY_Locked – if locked
- kStatus_GLIKEY_NotLocked – if unlocked

Returns

Status

status_t GLIKEY_CheckLock(GLIKEY_Type *base)

Check if Glikey is locked.

This operation returns the locking status of Glikey.

Return values

- kStatus_GLIKEY_LockedError – if locked
- kStatus_GLIKEY_NotLocked – if unlocked

Returns

Status kStatus_Success if success

status_t GLIKEY_SyncReset(GLIKEY_Type *base)

Perform a synchronous reset of Glikey.

This function performs a synchrounous reset of the Glikey. This results in:

- Glikey will return to the INIT state, unless it is in the LOCK state

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_SetIntEnable(GLIKEY_Type *base, uint32_t value)

Set interrupt enable flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance
- value – **[in]** Value to set the interrupt enable flag to, see #[TODO: add reference to constants]

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_GetIntEnable(GLIKEY_Type *base, uint32_t *value)

Get interrupt enable flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance
- value – **[out]** Pointer which will be filled with the interrupt enable status, see #[TODO: add reference to constants]

Returns

Status kStatus_Success if success

status_t GLIKEY_ClearIntStatus(GLIKEY_Type *base)

Clear the interrupt status flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_SetIntStatus(GLIKEY_Type *base)

Set the interrupt status flag of Glikey.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError

status_t GLIKEY_Lock(GLIKEY_Type *base)

Lock Glikey SFR (Special Function Registers) interface.

This operation locks the Glikey SFR interface if it is not locked yet.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success

status_t GLIKEY_LockIndex(GLIKEY_Type *base)

Lock Glikey index.

This operation is used to lock a Glikey index. It can only be executed from the WR_EN state, executing it from any other state will result in Glikey entering WR_DIS state. When this happens Glikey requires a reset (synchrous or asynchronous) to go back to INIT state. If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError, kStatus_GLIKEY_DisabledError

status_t GLIKEY_IsIndexLocked(GLIKEY_Type *base, uint32_t index)

Check if Glikey index is locked.

This operation returns the locking status of Glikey index.

Parameters

- base – **[in]** The base address of the Glikey instance
- index – **[in]** The index of the Glikey instance

Returns

kStatus_GLIKEY_Locked if locked, kStatus_GLIKEY_NotLocked if unlocked Possible errors: kStatus_Fail

status_t GLIKEY_StartEnable(GLIKEY_Type *base, uint32_t index)

Start Glikey enable.

This operation is used to set a new index and start a the sequence to enable it. It needs to be started from the INIT state. If the new index is already locked Glikey will go to LOCKED state, otherwise it will go to STEP1 state. If this operation is used when Glikey is in any state other than INIT Glikey will go to WR_DIS state. It can only recover from this state through a reset (synchrounous or asyncrhonous). If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance
- index – **[in]** The index of the Glikey instance

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError, kStatus_Fail

status_t GLIKEY_ContinueEnable(GLIKEY_Type *base, uint32_t codeword)

Continue Glikey enable.

This operation is used to progress through the different states of the state machine, starting from STEP1 until the state WR_EN is reached. Each next state of the state machine can only be reached by providing the right codeword to this function. If anything goes wrong the state machine will go to WR_DIS state and can only recover from it through a reset (synchrous or asynchronous). If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance
- codeword – **[in]** Encoded word for progressing to next FSM state (see GLIKEY_CODEWORD_STEPx/EN)

Returns

Status kStatus_Success if success Possible errors: kStatus_GLIKEY_LockedError, kStatus_Fail, kStatus_GLIKEY_DisabledError

status_t GLIKEY_EndOperation(GLIKEY_Type *base)

End Glikey operation.

This operation is used to end a Glikey operation. It can only be executed from the WR_EN, LOCKED and RESET states. Executing it from any other state will result in Glikey entering WR_DIS state. When this happens Glikey requires a reset (synchrous or asynchronous) to go back to INIT state. After this operation Glikey will go to INIT state or stay in LOCKED state when the index was locked. If the Glikey SFR lock is active this operation will return an error.

Parameters

- base – **[in]** The base address of the Glikey instance

Returns

A code-flow protected error code (see nxpCsslFlowProtection)

Returns

Status kStatus_Success if success, kStatus_GLIKEY_Locked if index is still locked Possible errors: kStatus_GLIKEY_LockedError, kStatus_GLIKEY_DisabledError

status_t GLIKEY_ResetIndex(GLIKEY_Type *base, uint32_t index)

Reset Glikey index.

This operation is used to reset a Glikey index. It can only be executed from the INIT state, executing it from any other state will result in Glikey entering WR_DIS state. When this happens Glikey requires a reset (synchrous or asynchronous) to go back to INIT state. If the Glikey SFR lock is active or the index is locked this operation will return an error.

Returns
A code-flow protected error code (see nxpCsslFlowProtection)

Returns
Status kStatus_Success if success, kStatus_GLIKEY_Locked if index is still locked Possible errors: kStatus_GLIKEY_LockedError, kStatus_GLIKEY_DisabledError

2.30 GPIO: General-Purpose Input/Output Driver

FSL_GPIO_DRIVER_VERSION
GPIO driver version.

enum _gpio_pin_direction
GPIO direction definition.

Values:

enumerator kGPIO_DigitalInput
Set current pin as digital input

enumerator kGPIO_DigitalOutput
Set current pin as digital output

enum _gpio_checker_attribute
GPIO checker attribute.

Values:

enumerator kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW
User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW
User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW
User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW
User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW
User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW
User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR
User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN
User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator kGPIO_IgnoreAttributeCheck
Ignores the attribute check

enum _gpio_interrupt_config
Configures the interrupt generation condition.
Values:

enumerator kGPIO_InterruptStatusFlagDisabled
Interrupt status flag is disabled.

enumerator kGPIO_DMARisingEdge
ISF flag and DMA request on rising edge.

enumerator kGPIO_DMAFallingEdge
ISF flag and DMA request on falling edge.

enumerator kGPIO_DMAEitherEdge
ISF flag and DMA request on either edge.

enumerator kGPIO_FlagRisingEdge
Flag sets on rising edge.

enumerator kGPIO_FlagFallingEdge
Flag sets on falling edge.

enumerator kGPIO_FlagEitherEdge
Flag sets on either edge.

enumerator kGPIO_InterruptLogicZero
Interrupt when logic zero.

enumerator kGPIO_InterruptRisingEdge
Interrupt on rising edge.

enumerator kGPIO_InterruptFallingEdge
Interrupt on falling edge.

enumerator kGPIO_InterruptEitherEdge
Interrupt on either edge.

enumerator kGPIO_InterruptLogicOne
Interrupt when logic one.

enumerator kGPIO_ActiveHighTriggerOutputEnable
Enable active high-trigger output.

enumerator kGPIO_ActiveLowTriggerOutputEnable
Enable active low-trigger output.

enum _gpio_interrupt_selection
Configures the selection of interrupt/DMA request/trigger output.
Values:

enumerator kGPIO_InterruptOutput0
Interrupt/DMA request/trigger output 0.

enumerator kGPIO_InterruptOutput1
Interrupt/DMA request/trigger output 1.

enum gpio_pin_interrupt_control_t
GPIO pin and interrupt control.
Values:

enumerator kGPIO_PinControlNonSecure
Pin Control Non-Secure.

enumerator kGPIO_InterruptControlNonSecure
Interrupt Control Non-Secure.

enumerator kGPIO_PinControlNonPrivilege
Pin Control Non-Privilege.

enumerator kGPIO_InterruptControlNonPrivilege
Interrupt Control Non-Privilege.

typedef enum *_gpio_pin_direction* gpio_pin_direction_t
GPIO direction definition.

typedef enum *_gpio_checker_attribute* gpio_checker_attribute_t
GPIO checker attribute.

typedef struct *_gpio_pin_config* gpio_pin_config_t
The GPIO pin configuration structure.
Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

typedef enum *_gpio_interrupt_config* gpio_interrupt_config_t
Configures the interrupt generation condition.

typedef enum *_gpio_interrupt_selection* gpio_interrupt_selection_t

Configures the selection of interrupt/DMA request/trigger output.

typedef struct *_gpio_version_info* gpio_version_info_t

GPIO version information.

GPIO_FIT_REG(value)

struct _gpio_pin_config

#include <fsl_gpio.h> The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

Public Members

gpio_pin_direction_t pinDirection

GPIO direction, input or output

uint8_t outputLogic

Set a default output logic, which has no use in input

struct _gpio_version_info

#include <fsl_gpio.h> GPIO version information.

Public Members

uint16_t feature

Feature Specification Number.

uint8_t minor

Minor Version Number.

uint8_t major

Major Version Number.

2.31 GPIO Driver

void GPIO_PortInit(GPIO_Type *base)

Initializes the GPIO peripheral.

This function ungates the GPIO clock.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PortDenit(GPIO_Type *base)

Denitializes the GPIO peripheral.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const *gpio_pin_config_t* *config)

Initializes a GPIO pin used by the board.

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the *GPIO_PinInit()* function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalInput,
  0,
}
Define a digital output pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalOutput,
  0,
}
```

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO port pin number
- config – GPIO pin configuration pointer

void GPIO_GetVersionInfo(GPIO_Type *base, *gpio_version_info_t* *verInfo)

Get GPIO version information.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- verInfo – GPIO version information

static inline void GPIO_SecurePrivilegeLock(GPIO_Type *base, *gpio_pin_interrupt_control_t* mask)

lock or unlock secure privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – pin or interrupt macro

static inline void GPIO_EnablePinControlNonSecure(GPIO_Type *base, uint32_t mask)

Enable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisablePinControlNonSecure(GPIO_Type *base, uint32_t mask)

Disable Pin Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Enable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisablePinControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Disable Pin Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)
Enable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonSecure(GPIO_Type *base, uint32_t mask)
Disable Interrupt Control Non-Secure.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_EnableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Enable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_DisableInterruptControlNonPrivilege(GPIO_Type *base, uint32_t mask)
Disable Interrupt Control Non-Privilege.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputEnable(GPIO_Type *base, uint32_t mask)
Enable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortInputDisable(GPIO_Type *base, uint32_t mask)
Disable port input.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PinWrite(GPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the multiple GPIO pins to the logic 1 or 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number
- output – GPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void GPIO_PortSet(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortClear(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortToggle(GPIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple GPIO pins.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t pin)

Reads the current input value of the GPIO port.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number

Return values

GPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

static inline void GPIO_SetPinInterruptConfig(GPIO_Type *base, uint32_t pin, *gpio_interrupt_config_t* config)

Configures the gpio pin interrupt/DMA request.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- config – GPIO pin interrupt configuration.

- *kGPIO_InterruptStatusFlagDisabled*: Interrupt/DMA request disabled.
- *kGPIO_DMARisingEdge* : DMA request on rising edge(if the DMA requests exit).
- *kGPIO_DMAFallingEdge*: DMA request on falling edge(if the DMA requests exit).
- *kGPIO_DMAEitherEdge* : DMA request on either edge(if the DMA requests exit).
- *kGPIO_FlagRisingEdge* : Flag sets on rising edge(if the Flag states exit).
- *kGPIO_FlagFallingEdge* : Flag sets on falling edge(if the Flag states exit).
- *kGPIO_FlagEitherEdge* : Flag sets on either edge(if the Flag states exit).
- *kGPIO_InterruptLogicZero* : Interrupt when logic zero.
- *kGPIO_InterruptRisingEdge* : Interrupt on rising edge.
- *kGPIO_InterruptFallingEdge*: Interrupt on falling edge.
- *kGPIO_InterruptEitherEdge* : Interrupt on either edge.
- *kGPIO_InterruptLogicOne* : Interrupt when logic one.
- *kGPIO_ActiveHighTriggerOutputEnable* : Enable active high-trigger output (if the trigger states exit).
- *kGPIO_ActiveLowTriggerOutputEnable* : Enable active low-trigger output (if the trigger states exit).

static inline void GPIO_SetPinInterruptChannel(GPIO_Type *base, uint32_t pin, *gpio_interrupt_selection_t* selection)

Configures the gpio pin interrupt/DMA request/trigger output channel selection.

Parameters

- base – GPIO peripheral base pointer.
- pin – GPIO pin number.
- selection – GPIO pin interrupt output selection.
  - *kGPIO_InterruptOutput0*: Interrupt/DMA request/trigger output 0.
  - *kGPIO_InterruptOutput1* : Interrupt/DMA request/trigger output 1.

uint32_t GPIO_GpioGetInterruptFlags(GPIO_Type *base)

Read the GPIO interrupt status flags.

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on.)

Returns

The current GPIO's interrupt status flag. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

uint32_t GPIO_GpioGetInterruptChannelFlags(GPIO_Type *base, uint32_t channel)

Read the GPIO interrupt status flags based on selected interrupt channel(IRQS).

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on.)
- channel – '0' means selete interrupt channel 0, '1' means selete interrupt channel 1.

Returns

The current GPIO's interrupt status flag based on the selected interrupt channel. '1' means the related pin's flag is set, '0' means the related pin's flag not set. For example, the return value 0x00010001 means the pin 0 and 17 have the interrupt pending.

uint8_t GPIO_PinGetInterruptFlag(GPIO_Type *base, uint32_t pin)

Read individual pin's interrupt status flag.

Parameters

- base – GPIO peripheral base pointer. (GPIOA, GPIOB, GPIOC, and so on)
- pin – GPIO specific pin number.

Returns

The current selected pin's interrupt status flag.

void GPIO_GpioClearInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears GPIO pin interrupt status flags.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

void GPIO_GpioClearInterruptChannelFlags(GPIO_Type *base, uint32_t mask, uint32_t channel)

Clears GPIO pin interrupt status flags based on selected interrupt channel(IRQS).

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro
- channel – '0' means selete interrupt channel 0, '1' means selete interrupt channel 1.

void GPIO_PinClearInterruptFlag(GPIO_Type *base, uint32_t pin)

Clear GPIO individual pin's interrupt status flag.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on).
- pin – GPIO specific pin number.

static inline void GPIO_SetMultipleInterruptPinsConfig(GPIO_Type *base, uint32_t mask, *gpio_interrupt_config_t* config)

Sets the GPIO interrupt configuration in PCR register for multiple pins.

Parameters

- base – GPIO peripheral base pointer.
- mask – GPIO pin number macro.
- config – GPIO pin interrupt configuration.
  - *kGPIO_InterruptStatusFlagDisabled*: Interrupt disabled.
  - *kGPIO_DMARisingEdge* : DMA request on rising edge(if the DMA requests exit).
  - *kGPIO_DMAFallingEdge*: DMA request on falling edge(if the DMA requests exit).
  - *kGPIO_DMAEitherEdge* : DMA request on either edge(if the DMA requests exit).

- *kGPIO_FlagRisingEdge* : Flag sets on rising edge(if the Flag states exit).
- *kGPIO_FlagFallingEdge* : Flag sets on falling edge(if the Flag states exit).
- *kGPIO_FlagEitherEdge* : Flag sets on either edge(if the Flag states exit).
- *kGPIO_InterruptLogicZero* : Interrupt when logic zero.
- *kGPIO_InterruptRisingEdge* : Interrupt on rising edge.
- *kGPIO_InterruptFallingEdge*: Interrupt on falling edge.
- *kGPIO_InterruptEitherEdge* : Interrupt on either edge.
- *kGPIO_InterruptLogicOne* : Interrupt when logic one.
- *kGPIO_ActiveHighTriggerOutputEnable* : Enable active high-trigger output (if the trigger states exit).
- *kGPIO_ActiveLowTriggerOutputEnable* : Enable active low-trigger output (if the trigger states exit)..

void GPIO_CheckAttributeBytes(GPIO_Type *base, *gpio_checker_attribute_t* attribute)

brief The GPIO module supports a device-specific number of data ports, organized as 32-bit words/8-bit Bytes. Each 32-bit/8-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. If the GPIO module's GACR register organized as 32-bit words, the attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- attribute – GPIO checker attribute

GPIO_PortGetInterruptFlags(base)

GPIO_PortClearInterruptFlags(base, mask)

2.32 I3C: I3C Driver

FSL_I3C_DRIVER_VERSION

I3C driver version.

I3C status return codes.

Values:

enumerator kStatus_I3C_Busy

The master is already performing a transfer.

enumerator kStatus_I3C_Idle

The slave driver is idle.

enumerator kStatus_I3C_Nak

The slave device sent a NAK in response to an address.

enumerator kStatus_I3C_WriteAbort
The slave device sent a NAK in response to a write.

enumerator kStatus_I3C_Term
The master terminates slave read.

enumerator kStatus_I3C_HdrParityError
Parity error from DDR read.

enumerator kStatus_I3C_CrcError
CRC error from DDR read.

enumerator kStatus_I3C_ReadFifoError
Read from M/SRDATAB register when FIFO empty.

enumerator kStatus_I3C_WriteFifoError
Write to M/SWDATAB register when FIFO full.

enumerator kStatus_I3C_MsgError
Message SDR/DDR mismatch or read/write message in wrong state

enumerator kStatus_I3C_InvalidReq
Invalid use of request.

enumerator kStatus_I3C_Timeout
The module has stalled too long in a frame.

enumerator kStatus_I3C_SlaveCountExceed
The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

enumerator kStatus_I3C_IBIWon
The I3C slave event IBI or MR or HJ won the arbitration on a header address.

enumerator kStatus_I3C_OverrunError
Slave internal from-bus buffer/FIFO overrun.

enumerator kStatus_I3C_UnderrunError
Slave internal to-bus buffer/FIFO underrun

enumerator kStatus_I3C_UnderrunNak
Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kStatus_I3C_InvalidStart
Slave invalid start flag

enumerator kStatus_I3C_SdrParityError
SDR parity error

enumerator kStatus_I3C_S0S1Error
S0 or S1 error

enum _i3c_hdr_mode
I3C HDR modes.
Values:

enumerator kI3C_HDRModeNone

enumerator kI3C_HDRModeDDR

enumerator kI3C_HDRModeTSP

enumerator kI3C_HDRModeTSL

typedef enum *_i3c_hdr_mode* i3c_hdr_mode_t
I3C HDR modes.

typedef struct *_i3c_device_info* i3c_device_info_t
I3C device information.

I3C_RETRY_TIMES
Max loops to wait for I3C operation status complete.
This is the maximum number of loops to wait for I3C operation status complete. If set to 0, it will wait indefinitely.

I3C_MAX_DEVCNT

I3C_IBI_BUFF_SIZE

struct _i3c_device_info
#include <fsl_i3c.h> I3C device information.

Public Members

uint8_t dynamicAddr
Device dynamic address.

uint8_t staticAddr
Static address.

uint8_t dcr
Device characteristics register information.

uint8_t bcr
Bus characteristics register information.

uint16_t vendorID
: Device vendor ID(manufacture ID).

uint32_t partNumber
: Device part number info

uint16_t maxReadLength
: Maximum read length.

uint16_t maxWriteLength
: Maximum write length.

uint8_t hdrMode
: Support hdr mode, could be OR logic in i3c_hdr_mode.

2.33 I3C Common Driver

typedef struct *_i3c_config* i3c_config_t
: Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

  This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the *I3C_GetDefaultConfig()* function and pass a pointer to your configuration structure instance.

  The configuration structure can be made constant so it resides in flash.

uint32_t I3C_GetInstance(I3C_Type *base)
: Get which instance current I3C is used.

  Parameters

  - base – The I3C peripheral base address.

void I3C_GetDefaultConfig(*i3c_config_t* *config)
: Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.

  This function provides the following default configuration for I3C:

```
config->enableMaster                = kI3C_MasterCapable;
config->disableTimeout              = false;
config->hKeep                       = kI3C_MasterHighKeeperNone;
config->enableOpenDrainStop         = true;
config->enableOpenDrainHigh         = true;
config->baudRate_Hz.i2cBaud         = 400000U;
config->baudRate_Hz.i3cPushPullBaud = 12500000U;
config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
config->masterDynamicAddress        = 0x0AU;
config->slowClock_Hz                = 1000000U;
config->enableSlave                 = true;
config->vendorID                    = 0x11BU;
config->enableRandomPart            = false;
config->partNumber                  = 0;
config->dcr                         = 0;
config->bcr = 0;
```

(continues on next page)

(continued from previous page)

```
config->hdrMode             = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest       = false;
config->ignoreS0S1Error     = false;
config->offline             = false;
config->matchSlaveStartStop = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with *I3C_Init()*.

Parameters

- config – **[out]** User provided configuration structure for default values. Refer to *i3c_config_t*.

void I3C_Init(I3C_Type *base, const *i3c_config_t* *config, uint32_t sourceClock_Hz)

Initializes the I3C peripheral. This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

Parameters

- base – The I3C peripheral base address.
- config – User provided peripheral configuration. Use *I3C_GetDefaultConfig()* to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

struct _i3c_config

#include <fsl_i3c.h> Structure with settings to initialize the I3C module, could both initialize master and slave functionality.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the *I3C_GetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster

Enable master mode.

bool disableTimeout

Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep

High keeper mode setting.

bool enableOpenDrainStop

Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh

Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz
: Desired baud rate settings.

i3c_start_scl_delay_t startSclDelay
: I3C SCL delay after START.

i3c_start_scl_delay_t restartSclDelay
: I3C SCL delay after Repeated START.

uint8_t masterDynamicAddress
: Main master dynamic address configuration.

uint32_t maxWriteLength
: Maximum write length.

uint32_t maxReadLength
: Maximum read length.

bool enableSlave
: Whether to enable slave.

uint8_t staticAddr
: Static address.

uint16_t vendorID
: Device vendor ID(manufacture ID).

uint32_t partNumber
: Device part number info

uint8_t dcr
: Device characteristics register information.

uint8_t bcr
: Bus characteristics register information.

uint8_t hdrMode
: Support hdr mode, could be OR logic in enumeration:i3c_hdr_mode_t.

bool nakAllRequest
: Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error
: Whether to ignore S0/S1 error in SDR mode.

bool offline
: Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop

Whether to assert start/stop status only the time slave is addressed.

2.34 I3C Master Driver

void I3C_MasterGetDefaultConfig(*i3c_master_config_t* *masterConfig)

Provides a default configuration for the I3C master peripheral.

This function provides the following default configuration for the I3C master peripheral:

```
masterConfig->enableMaster            = kI3C_MasterOn;
masterConfig->disableTimeout          = false;
masterConfig->hKeep                   = kI3C_MasterHighKeeperNone;
masterConfig->enableOpenDrainStop     = true;
masterConfig->enableOpenDrainHigh     = true;
masterConfig->baudRate_Hz             = 100000U;
masterConfig->busType                 = kI3C_TypeI2C;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with *I3C_MasterInit()*.

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to *i3c_master_config_t*.

void I3C_MasterInit(I3C_Type *base, const *i3c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the I3C master peripheral.

This function enables the peripheral clock and initializes the I3C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The I3C peripheral base address.
- masterConfig – User provided peripheral configuration. Use *I3C_MasterGetDefaultConfig()* to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void I3C_MasterDeinit(I3C_Type *base)

Deinitializes the I3C master peripheral.

This function disables the I3C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The I3C peripheral base address.

static inline void I3C_MasterEnable(I3C_Type *base, *i3c_master_enable_t* enable)

Set I3C module master mode.

Parameters

- base – The I3C peripheral base address.
- enable – Enable master mode.

status_t I3C_MasterCheckAndClearError(I3C_Type *base, uint32_t status)

status_t I3C_MasterWaitForCtrlDone(I3C_Type *base, bool waitIdle)

status_t I3C_CheckForBusyBus(I3C_Type *base)

void I3C_SlaveGetDefaultConfig(*i3c_slave_config_t* *slaveConfig)

Provides a default configuration for the I3C slave peripheral.

This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave              = true;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with I3C_SlaveInit().

Parameters

- slaveConfig – **[out]** User provided configuration structure for default values. Refer to *i3c_slave_config_t*.

void I3C_SlaveInit(I3C_Type *base, const *i3c_slave_config_t* *slaveConfig, uint32_t slowClock_Hz)

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- base – The I3C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use I3C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- slowClock_Hz – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH defines as 1, this parameter is useless.

void I3C_SlaveDeinit(I3C_Type *base)

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- base – The I3C peripheral base address.

static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnable)

Enable/Disable Slave.

Parameters

- base – The I3C peripheral base address.
- isEnable – Enable or disable.

static inline uint32_t I3C_MasterGetStatusFlags(I3C_Type *base)

Gets the I3C master status flags.

A bit mask with the state of all I3C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_master_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_MasterClearStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C master status flag state.

The following status register flags can be cleared:

- *kI3C_MasterSlaveStartFlag*
- *kI3C_MasterControlDoneFlag*
- *kI3C_MasterCompleteFlag*
- *kI3C_MasterArbitrationWonFlag*
- *kI3C_MasterSlave2MasterFlag*

Attempts to clear other flags has no effect.

See also:

_i3c_master_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of *_i3c_master_flags* enumerators OR'd together. You may pass the result of a previous call to *I3C_MasterGetStatusFlags()*.

static inline uint32_t I3C_MasterGetErrorStatusFlags(I3C_Type *base)

Gets the I3C master error status flags.

A bit mask with the state of all I3C master error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_master_error_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_MasterClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C master error status flag state.

See also:

_i3c_master_error_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of error status flags that are to be cleared. The mask is composed of *_i3c_master_error_flags* enumerators OR'd together. You may pass the result of a previous call to *I3C_MasterGetStatusFlags()*.

i3c_master_state_t I3C_MasterGetState(I3C_Type *base)

Gets the I3C master state.

Parameters

- base – The I3C peripheral base address.

Returns

I3C master state.

static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_slave_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*

Attempts to clear other flags has no effect.

See also:

_i3c_slave_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of *_i3c_slave_flags* enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetStatusFlags().

static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_slave_error_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C slave error status flag state.

See also:

_i3c_slave_error_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of error status flags that are to be cleared. The mask is composed of *_i3c_slave_error_flags* enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetErrorStatusFlags().

i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)

Gets the I3C slave state.

Parameters

- base – The I3C peripheral base address.

Returns

I3C slave activity state, refer *i3c_slave_activity_state_t*.

status_t I3C_SlaveCheckAndClearError(I3C_Type *base, uint32_t status)

static inline void I3C_MasterEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C master interrupt requests.

All flags except *kI3C_MasterBetweenFlag* and *kI3C_MasterNackDetectFlag* can be enabled as interrupts.

Parameters

- base – The I3C peripheral base address.

- interruptMask – Bit mask of interrupts to enable. See *_i3c_master_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_MasterDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C master interrupt requests.

All flags except *kI3C_MasterBetweenFlag* and *kI3C_MasterNackDetectFlag* can be enabled as interrupts.

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See *_i3c_master_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_MasterGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C master interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of *_i3c_master_flags* enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_MasterGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C master interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of *_i3c_master_flags* enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*
- *kI3C_SlaveRxReadyFlag*
- *kI3C_SlaveTxReadyFlag*
- *kI3C_SlaveDynamicAddrChangedFlag*
- *kI3C_SlaveReceivedCCCFlag*
- *kI3C_SlaveErrorFlag*
- *kI3C_SlaveHDRCommandMatchFlag*
- *kI3C_SlaveCCCHandledFlag*
- *kI3C_SlaveEventSentFlag*

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See *_i3c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*
- *kI3C_SlaveRxReadyFlag*
- *kI3C_SlaveTxReadyFlag*
- *kI3C_SlaveDynamicAddrChangedFlag*
- *kI3C_SlaveReceivedCCCFlag*
- *kI3C_SlaveErrorFlag*
- *kI3C_SlaveHDRCommandMatchFlag*
- *kI3C_SlaveCCCHandledFlag*
- *kI3C_SlaveEventSentFlag*

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See *_i3c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of *_i3c_slave_flags* enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of *_i3c_slave_flags* enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_MasterEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)

Enables or disables I3C master DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

static inline uint32_t I3C_MasterGetTxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C master transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Transmit Data Register address.

static inline uint32_t I3C_MasterGetRxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C master receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Master Receive Data Register address.

static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)

Enables or disables I3C slave DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

static inline void I3C_MasterSetWatermarks(I3C_Type *base, *i3c_tx_trigger_level_t* txLvl, *i3c_rx_trigger_level_t* rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C master FIFOs.

Parameters

- base – The I3C peripheral base address.
- txLvl – Transmit FIFO watermark level. The *kI3C_MasterTxReadyFlag* flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- rxLvl – Receive FIFO watermark level. The *kI3C_MasterRxReadyFlag* flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- flushTx – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- flushRx – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C_MasterGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of bytes in the I3C master FIFOs.

Parameters

- base – The I3C peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

static inline void I3C_SlaveSetWatermarks(I3C_Type *base, *i3c_tx_trigger_level_t* txLvl, *i3c_rx_trigger_level_t* rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.
- txLvl – Transmit FIFO watermark level. The *kI3C_SlaveTxReadyFlag* flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- rxLvl – Receive FIFO watermark level. The *kI3C_SlaveRxReadyFlag* flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- flushTx – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- flushRx – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

void I3C_MasterSetBaudRate(I3C_Type *base, const *i3c_baudrate_hz_t* *baudRate_Hz, uint32_t sourceClock_Hz)

Sets the I3C bus frequency for master transactions.

The I3C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: On devices affected by I3C errata ERR053429, an SDR read with PPBAUD < 2 can lose the last received byte when the RxFIFO fills before the CPU/DMA services it. For safe SDR reads, request a push-pull baud that yields PPBAUD > 1, or guarantee the CPU/DMA drains the RxFIFO fast enough to avoid a Full event.

Parameters

- base – The I3C peripheral base address.
- baudRate_Hz – Pointer to structure of requested bus frequency in Hertz.
- sourceClock_Hz – I3C functional clock frequency in Hertz.

status_t I3C_MasterGetI2CPresetTiming(*i3c_legacy_i2c_speed_mode_t* speed, *i3c_master_src_clk_t* srcClk, *i3c_timing_t* *timing)

Get prevalidated timing values for supported I2C speed.

Parameters

- speed – **[in]** I2C Speed mode.
- srcClk – **[in]** FCLK source clock.
- timing – **[out]** Prevalidated timing values. Valid when kStatus_Success is returned.

Returns

kStatus_Success A supported preset timing is found. kStatus_Fail The requested timing is not supported.

status_t I3C_MasterCalcTiming(const *i3c_master_baudrate_config_t* *baudRate, uint32_t sourceClock_Hz, *i3c_timing_t* *timing)

Calculate timing from desired baud rates.

Parameters

- baudRate – **[in]** Desired baud rates and bus configuration.
- sourceClock_Hz – **[in]** Master peripheral clock (FCLK) frequency in Hz.
- timing – **[out]** I3C timing register field values. Valid only when kStatus_Success is returned.

Returns

kStatus_Success Target rate is achieved. kStatus_Fail Target rate isn't achieved.

void I3C_MasterSetTiming(I3C_Type *base, const *i3c_timing_t* *timing)

Set I3C timing parameters.

Parameters

- base – I3C peripheral base address.
- timing – Pre-calculated timing values.

static inline bool I3C_MasterGetBusIdleState(I3C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The I3C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t I3C_MasterStartWithRxSize(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir, uint8_t rxSize)

Sends a START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the a address parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The I3C peripheral base address.
- type – The bus type to use in this transaction.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either *kI3C_Read* or *kI3C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- rxSize – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.

status_t I3C_MasterStart(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir)

Sends a START signal and slave address on the I2C/I3C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The I3C peripheral base address.
- type – The bus type to use in this transaction.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either *kI3C_Read* or *kI3C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.

status_t I3C_MasterRepeatedStartWithRxSize(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir, uint8_t rxSize)

Sends a repeated START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.

This function is used to send a Repeated START signal when a transfer is already in progress. Like *I3C_MasterStart()*, it also sends the specified 7-bit address. Call this API also configures the read terminate size for the following read transfer. For example, set the rxSize = 2, the following read transfer will be terminated after two bytes of data received. Write transfer will not be affected by the rxSize configuration.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The I3C peripheral base address.
- type – The bus type to use in this transaction.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either *kI3C_Read* or *kI3C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
- rxSize – Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.

static inline *status_t* I3C_MasterRepeatedStart(I3C_Type *base, *i3c_bus_type_t* type, uint8_t address, *i3c_direction_t* dir)

Sends a repeated START signal and slave address on the I2C/I3C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like *I3C_MasterStart()*, it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The I3C peripheral base address.
- type – The bus type to use in this transaction.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either *kI3C_Read* or *kI3C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.

status_t I3C_MasterSend(I3C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)

Performs a polling send transfer on the I2C/I3C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns *kStatus_I3C_Nak*.

Parameters

- base – The I3C peripheral base address.

- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- flags – Bit mask of options for the transfer. See enumeration *_i3c_master_transfer_flags* for available options.

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_Nak – The slave device sent a NAK in response to an address.
- kStatus_I3C_WriteAbort – The slave device sent a NAK in response to a write.
- kStatus_I3C_MsgError – Message SDR/DDR mismatch or read/write message in wrong state.
- kStatus_I3C_WriteFifoError – Write to M/SWDATAB register when FIFO full.
- kStatus_I3C_InvalidReq – Invalid use of request.

status_t I3C_MasterReceive(I3C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)

Performs a polling receive transfer on the I2C/I3C bus.

Parameters

- base – The I3C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- flags – Bit mask of options for the transfer. See enumeration *_i3c_master_transfer_flags* for available options.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_Term – The master terminates slave read.
- kStatus_I3C_HdrParityError – Parity error from DDR read.
- kStatus_I3C_CrcError – CRC error from DDR read.
- kStatus_I3C_MsgError – Message SDR/DDR mismatch or read/write message in wrong state.
- kStatus_I3C_ReadFifoError – Read from M/SRDATAB register when FIFO empty.
- kStatus_I3C_InvalidReq – Invalid use of request.

status_t I3C_MasterStop(I3C_Type *base)

Sends a STOP signal on the I2C/I3C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- base – The I3C peripheral base address.

Return values

- kStatus_Success – The STOP signal was successfully sent on the bus and the transaction terminated.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_InvalidReq – Invalid use of request.

void I3C_MasterEmitRequest(I3C_Type *base, *i3c_bus_request_t* masterReq)

I3C master emit request.

Parameters

- base – The I3C peripheral base address.
- masterReq – I3C master request of type *i3c_bus_request_t*

static inline void I3C_MasterEmitIBIResponse(I3C_Type *base, *i3c_ibi_response_t* ibiResponse)

I3C master emit request.

Parameters

- base – The I3C peripheral base address.
- ibiResponse – I3C master emit IBI response of type *i3c_ibi_response_t*

void I3C_MasterRegisterIBI(I3C_Type *base, *i3c_register_ibi_addr_t* *ibiRule)

I3C master register IBI rule.

Parameters

- base – The I3C peripheral base address.
- ibiRule – Pointer to ibi rule description of type *i3c_register_ibi_addr_t*

void I3C_MasterGetIBIRules(I3C_Type *base, *i3c_register_ibi_addr_t* *ibiRule)

I3C master get IBI rule.

Parameters

- base – The I3C peripheral base address.
- ibiRule – Pointer to store the read out ibi rule description.

i3c_ibi_type_t I3C_GetIBIType(I3C_Type *base)

I3C master get IBI Type.

Parameters

- base – The I3C peripheral base address.

Return values

i3c_ibi_type_t – Type of *i3c_ibi_type_t*.

static inline uint8_t I3C_GetIBIAddress(I3C_Type *base)

I3C master get IBI Address.

Parameters

- base – The I3C peripheral base address.

Return values

The – 8-bit IBI address.

status_t I3C_MasterProcessDAASpecifiedBaudrate(I3C_Type *base, uint8_t *addressList, uint32_t count, *i3c_master_daa_baudrate_t* *daaBaudRate)

Performs a DAA in the i3c bus with specified temporary baud rate.

Parameters

- base – The I3C peripheral base address.
- addressList – The pointer for address list which is used to do DAA.
- count – The address count in the address list.
- daaBaudRate – The temporary baud rate in DAA process, NULL for using initial setting. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- kStatus_I3C_SlaveCountExceed – The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

static inline *status_t* I3C_MasterProcessDAA(I3C_Type *base, uint8_t *addressList, uint32_t count)

Performs a DAA in the i3c bus.

Parameters

- base – The I3C peripheral base address.
- addressList – The pointer for address list which is used to do DAA.
- count – The address count in the address list. The initial setting is set back between the completion of the DAA and the return of this function.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
- kStatus_I3C_SlaveCountExceed – The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

i3c_device_info_t *I3C_MasterGetDeviceListAfterDAA(I3C_Type *base, uint8_t *count)

Get device information list after DAA process is done.

Parameters

- base – The I3C peripheral base address.
- count – **[out]** The pointer to store the available device count.

Returns

Pointer to the i3c_device_info_t array.

void I3C_MasterClearDeviceCount(I3C_Type *base)

Clear the global device count which represents current devices number on the bus. When user resets all dynamic addresses on the bus, should call this API.

Parameters

- base – The I3C peripheral base address.

status_t I3C_MasterTransferBlocking(I3C_Type *base, *i3c_master_transfer_t* *transfer)

Performs a master polling transfer on the I2C/I3C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- base – The I3C peripheral base address.
- transfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_I3C_Busy – Another master is currently utilizing the bus.
- kStatus_I3C_IBIWon – The I3C slave event IBI or MR or HJ won the arbitration on a header address.
- kStatus_I3C_Timeout – The module has stalled too long in a frame.
- kStatus_I3C_Nak – The slave device sent a NAK in response to an address.
- kStatus_I3C_WriteAbort – The slave device sent a NAK in response to a write.
- kStatus_I3C_Term – The master terminates slave read.
- kStatus_I3C_HdrParityError – Parity error from DDR read.
- kStatus_I3C_CrcError – CRC error from DDR read.
- kStatus_I3C_MsgError – Message SDR/DDR mismatch or read/write message in wrong state.
- kStatus_I3C_ReadFifoError – Read from M/SRDATAB register when FIFO empty.
- kStatus_I3C_WriteFifoError – Write to M/SWDATAB register when FIFO full.
- kStatus_I3C_InvalidReq – Invalid use of request.

status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)

Performs a polling send transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

void I3C_MasterTransferCreateHandle(I3C_Type *base, *i3c_master_handle_t* *handle, const *i3c_master_transfer_callback_t* *callback, void *userData)

Creates a new handle for the I3C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the *I3C_MasterTransferAbort()* API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The I3C peripheral base address.
- handle – **[out]** Pointer to the I3C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t I3C_MasterTransferNonBlocking(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C/I3C bus.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t I3C_MasterTransferGetCount(I3C_Type *base, *i3c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void I3C_MasterTransferAbort(I3C_Type *base, *i3c_master_handle_t* *handle)

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.

void I3C_MasterTransferHandleIRQ(I3C_Type *base, void *intHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- intHandle – Pointer to the I3C master driver handle.

void I3C_DriverIRQHandler(uint32_t instance)

Common IRQ handler entry for all I3C instances.

This function dispatches the IRQ to the correct instance handler using the instance index.

Parameters

- instance – I3C instance number.

enum _i3c_master_flags

I3C master peripheral flags.

The following status register flags can be cleared:

- *kI3C_MasterSlaveStartFlag*
- *kI3C_MasterControlDoneFlag*
- *kI3C_MasterCompleteFlag*
- *kI3C_MasterArbitrationWonFlag*
- *kI3C_MasterSlave2MasterFlag*

All flags except *kI3C_MasterBetweenFlag* and *kI3C_MasterNackDetectFlag* can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterBetweenFlag

Between messages/DAAs flag

enumerator kI3C_MasterNackDetectFlag

NACK detected flag

enumerator kI3C_MasterSlaveStartFlag
Slave request start flag

enumerator kI3C_MasterControlDoneFlag
Master request complete flag

enumerator kI3C_MasterCompleteFlag
Transfer complete flag

enumerator kI3C_MasterRxReadyFlag
Rx data ready in Rx buffer flag

enumerator kI3C_MasterTxReadyFlag
Tx buffer ready for Tx data flag

enumerator kI3C_MasterArbitrationWonFlag
Header address won arbitration flag

enumerator kI3C_MasterErrorFlag
Error occurred flag

enumerator kI3C_MasterSlave2MasterFlag
Switch from slave to master flag

enumerator kI3C_MasterClearFlags

enum _i3c_master_error_flags
I3C master error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_MasterErrorNackFlag
Slave NACKed the last address

enumerator kI3C_MasterErrorWriteAbortFlag
Slave NACKed the write data

enumerator kI3C_MasterErrorParityFlag
Parity error from DDR read

enumerator kI3C_MasterErrorCrcFlag
CRC error from DDR read

enumerator kI3C_MasterErrorReadFlag
Read from MRDATAB register when FIFO empty

enumerator kI3C_MasterErrorWriteFlag
Write to MWDATAB register when FIFO full

enumerator kI3C_MasterErrorMsgFlag
Message SDR/DDR mismatch or read/write message in wrong state

enumerator kI3C_MasterErrorInvalidReqFlag
Invalid use of request

enumerator kI3C_MasterErrorTimeoutFlag
The module has stalled too long in a frame

enumerator kI3C_MasterAllErrorFlags
All error flags

enum _i3c_master_state
I3C working master state.
Values:

enumerator kI3C_MasterStateIdle
Bus stopped.

enumerator kI3C_MasterStateSlvReq
Bus stopped but slave holding SDA low.

enumerator kI3C_MasterStateMsgSdr
In SDR Message mode from using MWMSG_SDR.

enumerator kI3C_MasterStateNormAct
In normal active SDR mode.

enumerator kI3C_MasterStateDdr
In DDR Message mode.

enumerator kI3C_MasterStateDaa
In ENTDAA mode.

enumerator kI3C_MasterStateIbiAck
Waiting on IBI ACK/NACK decision.

enumerator kI3C_MasterStateIbiRcv
Receiving IBI.

enum _i3c_master_enable
I3C master enable configuration.
Values:

enumerator kI3C_MasterOff
Master off.

enumerator kI3C_MasterOn
Master on.

enumerator kI3C_MasterCapable
Master capable.

enum _i3c_master_hkeep
I3C high keeper configuration.
Values:

enumerator kI3C_MasterHighKeeperNone
Use PUR to hold SCL high.

enumerator kI3C_MasterHighKeeperWiredIn
Use pin_HK controls.

enumerator kI3C_MasterPassiveSDA
Hi-Z for Bus Free and hold SDA.

enumerator kI3C_MasterPassiveSDASCL
Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

enum _i3c_bus_request
Emits the requested operation when doing in pieces vs. by message.
Values:

enumerator kI3C_RequestNone
No request.

enumerator kI3C_RequestEmitStartAddr
Request to emit start and address on bus.

enumerator kI3C_RequestEmitStop
Request to emit stop on bus.

enumerator kI3C_RequestIbiAckNack
Manual IBI ACK or NACK.

enumerator kI3C_RequestProcessDAA
Process DAA.

enumerator kI3C_RequestForceExit
Request to force exit.

enumerator kI3C_RequestAutoIbi
Hold in stopped state, but Auto-emit START,7E.

enum _i3c_bus_type
Bus type with EmitStartAddr.
Values:

enumerator kI3C_TypeI3CSdr
SDR mode of I3C.

enumerator kI3C_TypeI2C
Standard i2c protocol.

enumerator kI3C_TypeI3CDdr
HDR-DDR mode of I3C.

enum _i3c_ibi_response
IBI response.
Values:

enumerator kI3C_IbiRespAck
ACK with no mandatory byte.

enumerator kI3C_IbiRespNack
NACK.

enumerator kI3C_IbiRespAckMandatory
ACK with mandatory byte.

enumerator kI3C_IbiRespManual
Reserved.

enum _i3c_ibi_type
IBI type.
Values:

enumerator kI3C_IbiNormal
In-band interrupt.

enumerator kI3C_IbiHotJoin
slave hot join.

enumerator kI3C_IbiMasterRequest
slave master ship request.

enum _i3c_ibi_state
IBI state.
Values:

enumerator kI3C_IbiReady
In-band interrupt ready state, ready for user to handle.

enumerator kI3C_IbiDataBuffNeed
In-band interrupt need data buffer for data receive.

enumerator kI3C_IbiAckNackPending
In-band interrupt Ack/Nack pending for decision.

enum _i3c_direction
Direction of master and slave transfers.
Values:

enumerator kI3C_Write
Master transmit.

enumerator kI3C_Read
Master receive.

enum _i3c_tx_trigger_level
Watermark of TX int/dma trigger level.
Values:

enumerator kI3C_TxTriggerOnEmpty
Trigger on empty.

enumerator kI3C_TxTriggerUntilOneQuarterOrLess
Trigger on 1/4 full or less.

enumerator kI3C_TxTriggerUntilOneHalfOrLess
Trigger on 1/2 full or less.

enumerator kI3C_TxTriggerUntilOneLessThanFull
Trigger on 1 less than full or less.

enum _i3c_rx_trigger_level
Watermark of RX int/dma trigger level.
Values:

enumerator kI3C_RxTriggerOnNotEmpty
Trigger on not empty.

enumerator kI3C_RxTriggerUntilOneQuarterOrMore
Trigger on 1/4 full or more.

enumerator kI3C_RxTriggerUntilOneHalfOrMore
Trigger on 1/2 full or more.

enumerator kI3C_RxTriggerUntilThreeQuarterOrMore
Trigger on 3/4 full or more.

enum _i3c_rx_term_ops
I3C master read termination operations.
Values:

enumerator kI3C_RxTermDisable
Master doesn't terminate read, used for CCC transfer.

enumerator kI3C_RxAutoTerm
Master auto terminate read after receiving specified bytes(<=255).

enumerator kI3C_RxTermLastByte
Master terminates read at any time after START, no length limitation.

enum _i3c_start_scl_delay
I3C start SCL delay options.
Values:

enumerator kI3C_NoDelay
No delay.

enumerator kI3C_IncreaseSclHalfPeriod
Increases SCL clock period by 1/2.

enumerator kI3C_IncreaseSclOnePeriod
Increases SCL clock period by 1.

enumerator kI3C_IncreaseSclOneAndHalfPeriod
Increases SCL clock period by 1 1/2

enum _i3c_master_bus_mode
I3C bus topology. Selects how the baud rate is calculated for the bus.
Values:

enumerator kI3C_PureBus
Only I3C devices on the bus.

enumerator kI3C_MixedFastBus
I3C and legacy I2C Fm/Fm+ devices share the bus.

enumerator kI3C_LimitedBus
I3C controller on a legacy I2C bus.

enum _i3c_legacy_i2c_speed_mode
Legacy I2C preset selector.
Values:

enumerator kI3C_LegacyI2CStandardMode

Reserved for future Standard-mode support.

enumerator kI3C_LegacyI2CFastMode

Legacy I2C Fast-mode preset.

enumerator kI3C_LegacyI2CFastModePlus

Legacy I2C Fast-mode Plus preset.

enum _i3c_master_src_clk

Validated source clock selector for I2C timing presets.

Values:

enumerator kI3C_MasterSrcClk24M

Use the validated 24 MHz timing table.

enumerator kI3C_MasterSrcClk25M

Use the validated 25 MHz timing table.

enum _i3c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the *_i3c_master_transfer::flags* field.

Values:

enumerator kI3C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kI3C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kI3C_TransferRepeatedStartFlag

Send a repeated start condition

enumerator kI3C_TransferNoStopFlag

Don't send a stop condition.

enumerator kI3C_TransferWordsFlag

Transfer in words, else transfer in bytes.

enumerator kI3C_TransferDisableRxTermFlag

Disable Rx termination. Note: It's for I3C CCC transfer.

enumerator kI3C_TransferRxAutoTermFlag

Set Rx auto-termination. Note: It's adaptive based on Rx size(<=255 bytes) except in I3C_MasterReceive.

enumerator kI3C_TransferStartWithBroadcastAddr
Start transfer with 0x7E, then read/write data with device address.

typedef enum *_i3c_master_state* i3c_master_state_t
I3C working master state.

typedef enum *_i3c_master_enable* i3c_master_enable_t
I3C master enable configuration.

typedef enum *_i3c_master_hkeep* i3c_master_hkeep_t
I3C high keeper configuration.

typedef enum *_i3c_bus_request* i3c_bus_request_t
Emits the requested operation when doing in pieces vs. by message.

typedef enum *_i3c_bus_type* i3c_bus_type_t
Bus type with EmitStartAddr.

typedef enum *_i3c_ibi_response* i3c_ibi_response_t
IBI response.

typedef enum *_i3c_ibi_type* i3c_ibi_type_t
IBI type.

typedef enum *_i3c_ibi_state* i3c_ibi_state_t
IBI state.

typedef enum *_i3c_direction* i3c_direction_t
Direction of master and slave transfers.

typedef enum *_i3c_tx_trigger_level* i3c_tx_trigger_level_t
Watermark of TX int/dma trigger level.

typedef enum *_i3c_rx_trigger_level* i3c_rx_trigger_level_t
Watermark of RX int/dma trigger level.

typedef enum *_i3c_rx_term_ops* i3c_rx_term_ops_t
I3C master read termination operations.

typedef enum *_i3c_start_scl_delay* i3c_start_scl_delay_t
I3C start SCL delay options.

typedef struct *_i3c_register_ibi_addr* i3c_register_ibi_addr_t
Structure with setting master IBI rules and slave registry.

typedef struct *_i3c_baudrate* i3c_baudrate_hz_t
Structure with I3C baudrate settings.

typedef struct *_i3c_master_daa_baudrate* i3c_master_daa_baudrate_t

I3C DAA baud rate configuration.

typedef enum *_i3c_master_bus_mode* i3c_master_bus_mode_t

I3C bus topology. Selects how the baud rate is calculated for the bus.

typedef enum *_i3c_legacy_i2c_speed_mode* i3c_legacy_i2c_speed_mode_t

Legacy I2C preset selector.

typedef enum *_i3c_master_src_clk* i3c_master_src_clk_t

Validated source clock selector for I2C timing presets.

typedef struct *_i3c_master_baudrate_config* i3c_master_baudrate_config_t

I3C master baud rate configuration.

typedef struct *_i3c_timing* i3c_timing_t

Pre-calculated I3C MCONFIG timing fields.

Pre-compute the timing at init, then switch with *I3C_MasterSetTiming()*. The actualXxxBaud fields report the rates the hardware achieves.

typedef struct *_i3c_master_config* i3c_master_config_t

Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the *I3C_MasterGetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef struct *_i3c_master_transfer* i3c_master_transfer_t

typedef struct *_i3c_master_handle* i3c_master_handle_t

typedef struct *_i3c_master_transfer_callback* i3c_master_transfer_callback_t

i3c master callback functions.

typedef void (*i3c_master_isr_t)(I3C_Type *base, void *handle)

Typedef for master interrupt handler.

struct __i3c_register_ibi_addr

#include <fsl_i3c.h> Structure with setting master IBI rules and slave registry.

Public Members

uint8_t address[5]

Address array for registry.

bool i3cFastStart

Allow the START header to run as push-pull speed if all dynamic addresses take MSB 0.

bool ibiHasPayload

Whether the address array has mandatory IBI byte.

struct _i3c_baudrate

#include <fsl_i3c.h> Structure with I3C baudrate settings.

Public Members

uint32_t i2cBaud

Desired I2C baud rate in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct _i3c_master_daa_baudrate

#include <fsl_i3c.h> I3C DAA baud rate configuration.

Public Members

uint32_t sourceClock_Hz

FCLK, function clock in Hertz.

uint32_t i3cPushPullBaud

Desired I3C push-pull baud rate in Hertz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain baud rate in Hertz.

struct _i3c_master_baudrate_config

#include <fsl_i3c.h> I3C master baud rate configuration.

Public Members

i3c_master_bus_mode_t busMode

Bus topology.

uint32_t i2cBaud

Desired legacy I2C baud rate in Hz; 0 = do not program I2CBAUD.

uint32_t i3cPushPullBaud

Desired I3C push-pull SCL frequency in Hz.

uint32_t i3cOpenDrainBaud

Desired I3C open-drain SCL frequency in Hz.

bool enableODHPP

Open-drain HIGH shape: true = narrow (normal I3C frames), false = symmetric (required for the first-7Eh broadcast).

struct _i3c_timing

#include <fsl_i3c.h> Pre-calculated I3C MCONFIG timing fields.

Pre-compute the timing at init, then switch with *I3C_MasterSetTiming()*. The actualXxxBaud fields report the rates the hardware achieves.

Public Members

uint8_t ppBaud

MCONFIG[PPBAUD] field.

uint8_t ppLow

MCONFIG[PPLOW] field.

uint8_t odBaud

MCONFIG[ODBAUD] field (0 is forbidden).

bool odHighPP

MCONFIG[ODHPP] field.

uint8_t i2cBaud

MCONFIG[I2CBAUD] field.

uint32_t actualPpBaud

Achieved PP SCL frequency in Hz.

uint32_t actualOdBaud

Achieved OD SCL frequency in Hz.

uint32_t actualI2cBaud

Achieved I2C SCL frequency in Hz (0 if no I2C is configured).

struct _i3c_master_config

#include <fsl_i3c.h> Structure with settings to initialize the I3C master module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the *I3C_MasterGetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

i3c_master_enable_t enableMaster
Enable master mode.

bool disableTimeout
Whether to disable timeout to prevent the ERRWARN.

i3c_master_hkeep_t hKeep
High keeper mode setting.

bool enableOpenDrainStop
Whether to emit open-drain speed STOP.

bool enableOpenDrainHigh
Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.

i3c_baudrate_hz_t baudRate_Hz
Desired baud rate settings.

i3c_start_scl_delay_t startSclDelay
I3C SCL delay after START.

i3c_start_scl_delay_t restartSclDelay
I3C SCL delay after Repeated START.

struct __i3c_master_transfer_callback
#include <fsl_i3c.h> i3c master callback functions.

Public Members

void (*slave2Master)(I3C_Type *base, void *userData)
Transfer complete callback

void (*ibiCallback)(I3C_Type *base, *i3c_master_handle_t* *handle, *i3c_ibi_type_t* ibiType, *i3c_ibi_state_t* ibiState)
IBI event callback

void (*transferComplete)(I3C_Type *base, *i3c_master_handle_t* *handle, *status_t* completionStatus, void *userData)
Transfer complete callback

struct __i3c_master_transfer
#include <fsl_i3c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the *I3C_MasterTransferNonBlocking()* API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration *_i3c_master_transfer_flags* for available options. Set to 0 or *kI3C_TransferDefaultFlag* for normal transfers.

uint8_t slaveAddress

The 7-bit slave address.

i3c_direction_t direction

Either *kI3C_Read* or *kI3C_Write*.

uint32_t subaddress

Sub address. Transferred MSB first.

size_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data

Pointer to data to transfer.

size_t dataSize

Number of bytes to transfer.

i3c_bus_type_t busType

bus type.

i3c_ibi_response_t ibiResponse

ibi response during transfer.

struct __i3c_master_handle

#include <fsl_i3c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint32_t remainingBytes

Remaining byte count in current state.

i3c_rx_term_ops_t rxTermOps

Read termination operation.

i3c_master_transfer_t transfer
: Copy of the current transfer info.

uint8_t ibiAddress
: Slave address which request IBI.

uint8_t *ibiBuff
: Pointer to IBI buffer to keep ibi bytes.

size_t ibiPayloadSize
: IBI payload size.

i3c_ibi_type_t ibiType
: IBI type.

i3c_master_transfer_callback_t callback
: Callback functions pointer.

void *userData
: Application data passed to callback.

2.35 I3C Master DMA Driver

void I3C_MasterTransferCreateHandleEDMA(I3C_Type *base, *i3c_master_edma_handle_t* *handle, const *i3c_master_edma_callback_t* *callback, void *userData, *edma_handle_t* *rxDmaHandle, *edma_handle_t* *txDmaHandle)

Create a new handle for the I3C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I3C_MasterTransferAbortDMA() API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.
- rxDmaHandle – Handle for the DMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the DMA transmit channel. Created by the user prior to calling this function.

status_t I3C_MasterTransferEDMA(I3C_Type *base, *i3c_master_edma_handle_t* *handle, *i3c_master_transfer_t* *transfer)

Performs a non-blocking DMA-based transaction on the I3C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Note: On parts affected by ERR053412, the I3C controller stalls SCL when the RxFIFO becomes full and does not recover after CPU read FIFO, bus will be stalling indefinitely. This DMA-based API services the RxFIFO with EDMA so it never fills, and is the recommended workaround on affected parts.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

status_t I3C_MasterTransferGetCountEDMA(I3C_Type *base, *i3c_master_edma_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a DMA transaction currently in progress.

void I3C_MasterTransferAbortEDMA(I3C_Type *base, *i3c_master_edma_handle_t* *handle)

Terminates a non-blocking I3C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the DMA peripheral's IRQ priority.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C master driver handle.

void I3C_MasterTransferEDMAHandleIRQ(I3C_Type *base, void *i3cHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- i3cHandle – Pointer to the I3C master DMA driver handle.

typedef struct *_i3c_master_edma_handle* i3c_master_edma_handle_t

typedef struct *_i3c_master_edma_callback* i3c_master_edma_callback_t

i3c master callback functions.

struct i3c_master_edma_callback

#include <fsl_i3c_edma.h> i3c master callback functions.

Public Members

void (*slave2Master)(I3C_Type *base, void *userData)

Target asks for controller request.

void (*ibiCallback)(I3C_Type *base, *i3c_master_edma_handle_t* *handle, *i3c_ibi_type_t* ibiType, *i3c_ibi_state_t* ibiState)

IBI event callback.

void (*transferComplete)(I3C_Type *base, *i3c_master_edma_handle_t* *handle, *status_t* status, void *userData)

Transfer complete callback.

struct i3c_master_edma_handle

#include <fsl_i3c_edma.h> Driver handle for master EDMA APIs.

Note: The contents of this structure are private and subject to change.

Public Members

I3C_Type *base

I3C base pointer.

uint8_t state

Transfer state machine current state.

uint32_t transferCount

Indicates progress of the transfer

uint8_t subaddressBuffer[4]

Saving subaddress command.

uint8_t subaddressCount

Saving command count.

i3c_master_transfer_t transfer
: Copy of the current transfer info.

i3c_master_edma_callback_t callback
: Callback function pointer.

void *userData
: Application data passed to callback.

edma_handle_t *rxDmaHandle
: Handle for receive DMA channel.

edma_handle_t *txDmaHandle
: Handle for transmit DMA channel.

bool ibiFlag
: IBIWON flag.

uint8_t ibiAddress
: Slave address which request IBI.

uint8_t *ibiBuff
: Pointer to IBI buffer to keep ibi bytes.

size_t ibiPayloadSize
: IBI payload size.

i3c_ibi_type_t ibiType
: IBI type.

status_t result
: Transfer result.

2.36 I3C Slave Driver

void I3C_SlaveGetDefaultConfig(*i3c_slave_config_t* *slaveConfig)
: Provides a default configuration for the I3C slave peripheral.

  This function provides the following default configuration for the I3C slave peripheral:

```
slaveConfig->enableslave          = true;
```

  After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with I3C_SlaveInit().

  Parameters

  - slaveConfig – **[out]** User provided configuration structure for default values. Refer to *i3c_slave_config_t*.

void I3C_SlaveInit(I3C_Type *base, const *i3c_slave_config_t* *slaveConfig, uint32_t slowClock_Hz)

Initializes the I3C slave peripheral.

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

- base – The I3C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use I3C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
- slowClock_Hz – Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values. If FSL_FEATURE_I3C_HAS_NO_SCONFIG_BAMATCH defines as 1, this parameter is useless.

void I3C_SlaveDeinit(I3C_Type *base)

Deinitializes the I3C slave peripheral.

This function disables the I3C slave peripheral and gates the clock.

Parameters

- base – The I3C peripheral base address.

static inline void I3C_SlaveEnable(I3C_Type *base, bool isEnable)

Enable/Disable Slave.

Parameters

- base – The I3C peripheral base address.
- isEnable – Enable or disable.

static inline uint32_t I3C_SlaveGetStatusFlags(I3C_Type *base)

Gets the I3C slave status flags.

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_slave_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_SlaveClearStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C slave status flag state.

The following status register flags can be cleared:

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*

Attempts to clear other flags has no effect.

See also:

_i3c_slave_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of *_i3c_slave_flags* enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetStatusFlags().

static inline uint32_t I3C_SlaveGetErrorStatusFlags(I3C_Type *base)

Gets the I3C slave error status flags.

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i3c_slave_error_flags

Parameters

- base – The I3C peripheral base address.

Returns

State of the error status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void I3C_SlaveClearErrorStatusFlags(I3C_Type *base, uint32_t statusMask)

Clears the I3C slave error status flag state.

See also:

_i3c_slave_error_flags.

Parameters

- base – The I3C peripheral base address.
- statusMask – A bitmask of error status flags that are to be cleared. The mask is composed of *_i3c_slave_error_flags* enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetErrorStatusFlags().

i3c_slave_activity_state_t I3C_SlaveGetActivityState(I3C_Type *base)

Gets the I3C slave state.

Parameters

- base – The I3C peripheral base address.

Returns

I3C slave activity state, refer *i3c_slave_activity_state_t*.

static inline void I3C_SlaveEnableInterrupts(I3C_Type *base, uint32_t interruptMask)

Enables the I3C slave interrupt requests.

Only below flags can be enabled as interrupts.

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*
- *kI3C_SlaveRxReadyFlag*
- *kI3C_SlaveTxReadyFlag*
- *kI3C_SlaveDynamicAddrChangedFlag*
- *kI3C_SlaveReceivedCCCFlag*
- *kI3C_SlaveErrorFlag*
- *kI3C_SlaveHDRCommandMatchFlag*
- *kI3C_SlaveCCCHandledFlag*
- *kI3C_SlaveEventSentFlag*

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See *_i3c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline void I3C_SlaveDisableInterrupts(I3C_Type *base, uint32_t interruptMask)

Disables the I3C slave interrupt requests.

Only below flags can be disabled as interrupts.

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*
- *kI3C_SlaveRxReadyFlag*
- *kI3C_SlaveTxReadyFlag*
- *kI3C_SlaveDynamicAddrChangedFlag*
- *kI3C_SlaveReceivedCCCFlag*
- *kI3C_SlaveErrorFlag*
- *kI3C_SlaveHDRCommandMatchFlag*
- *kI3C_SlaveCCCHandledFlag*
- *kI3C_SlaveEventSentFlag*

Parameters

- base – The I3C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See *_i3c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I3C_SlaveGetEnabledInterrupts(I3C_Type *base)

Returns the set of currently enabled I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of *_i3c_slave_flags* enumerators OR'd together to indicate the set of enabled interrupts.

static inline uint32_t I3C_SlaveGetPendingInterrupts(I3C_Type *base)

Returns the set of pending I3C slave interrupt requests.

Parameters

- base – The I3C peripheral base address.

Returns

A bitmask composed of *_i3c_slave_flags* enumerators OR'd together to indicate the set of pending interrupts.

static inline void I3C_SlaveEnableDMA(I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)

Enables or disables I3C slave DMA requests.

Parameters

- base – The I3C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.
- width – DMA read/write unit in bytes.

static inline uint32_t I3C_SlaveGetTxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave transmit data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

static inline uint32_t I3C_SlaveGetRxFifoAddress(I3C_Type *base, uint32_t width)

Gets I3C slave receive data register address for DMA transfer.

Parameters

- base – The I3C peripheral base address.
- width – DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

static inline void I3C_SlaveSetWatermarks(I3C_Type *base, *i3c_tx_trigger_level_t* txLvl, *i3c_rx_trigger_level_t* rxLvl, bool flushTx, bool flushRx)

Sets the watermarks for I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.

- txLvl – Transmit FIFO watermark level. The *kI3C_SlaveTxReadyFlag* flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
- rxLvl – Receive FIFO watermark level. The *kI3C_SlaveRxReadyFlag* flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
- flushTx – true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
- flushRx – true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

static inline void I3C_SlaveGetFifoCounts(I3C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of bytes in the I3C slave FIFOs.

Parameters

- base – The I3C peripheral base address.
- txCount – **[out]** Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

status_t I3C_SlaveSend(I3C_Type *base, const void *txBuff, size_t txSize)

Performs a polling send transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

status_t I3C_SlaveReceive(I3C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I3C bus.

Parameters

- base – The I3C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

void I3C_SlaveTransferCreateHandle(I3C_Type *base, *i3c_slave_handle_t* *handle, *i3c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the I3C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the *I3C_SlaveTransferAbort()* API shall be called.

Note: The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The I3C peripheral base address.
- handle – **[out]** Pointer to the I3C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t I3C_SlaveTransferNonBlocking(I3C_Type *base, *i3c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and *I3C_SlaveTransferCreateHandle()* to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to *I3C_SlaveTransferCreateHandle()*. The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *i3c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kI3C_SlaveTransmitEvent* and *kI3C_SlaveReceiveEvent* events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kI3C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to struct: *_i3c_slave_handle* structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together *i3c_slave_transfer_event_t* enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and *kI3C_SlaveAllEvents* to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_I3C_Busy – Slave transfers have already been started on this handle.

status_t I3C_SlaveTransferGetCount(I3C_Type *base, *i3c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer status during a non-blocking transfer.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure.
- count – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress –

void I3C_SlaveTransferAbort(I3C_Type *base, *i3c_slave_handle_t* *handle)

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to struct: *_i3c_slave_handle* structure which stores the transfer state.

void I3C_SlaveTransferHandleIRQ(I3C_Type *base, void *intHandle)

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- intHandle – Pointer to struct: *_i3c_slave_handle* structure which stores the transfer state.

enum _i3c_slave_flags

I3C slave peripheral flags.

The following status register flags can be cleared:

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*

Only below flags can be enabled as interrupts.

- *kI3C_SlaveBusStartFlag*
- *kI3C_SlaveMatchedFlag*
- *kI3C_SlaveBusStopFlag*
- *kI3C_SlaveRxReadyFlag*
- *kI3C_SlaveTxReadyFlag*
- *kI3C_SlaveDynamicAddrChangedFlag*
- *kI3C_SlaveReceivedCCCFlag*
- *kI3C_SlaveErrorFlag*
- *kI3C_SlaveHDRCommandMatchFlag*
- *kI3C_SlaveCCCHandledFlag*
- *kI3C_SlaveEventSentFlag*

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_SlaveNotStopFlag

Slave status not stop flag

enumerator kI3C_SlaveMessageFlag
Slave status message, indicating slave is listening to the bus traffic or responding

enumerator kI3C_SlaveRequiredReadFlag
Slave status required, either is master doing SDR read from slave, or is IBI pushing out.

enumerator kI3C_SlaveRequiredWriteFlag
Slave status request write, master is doing SDR write to slave, except slave in ENTDAA mode

enumerator kI3C_SlaveBusDAAFlag
I3C bus is in ENTDAA mode

enumerator kI3C_SlaveBusHDRModeFlag
I3C bus is in HDR mode

enumerator kI3C_SlaveBusStartFlag
Start/Re-start event is seen since the bus was last cleared

enumerator kI3C_SlaveMatchedFlag
Slave address(dynamic/static) matched since last cleared

enumerator kI3C_SlaveBusStopFlag
Stop event is seen since the bus was last cleared

enumerator kI3C_SlaveRxReadyFlag
Rx data ready in rx buffer flag

enumerator kI3C_SlaveTxReadyFlag
Tx buffer ready for Tx data flag

enumerator kI3C_SlaveDynamicAddrChangedFlag
Slave dynamic address has been assigned, re-assigned, or lost

enumerator kI3C_SlaveReceivedCCCFlag
Slave received Common command code

enumerator kI3C_SlaveErrorFlag
Error occurred flag

enumerator kI3C_SlaveHDRCommandMatchFlag
High data rate command match

enumerator kI3C_SlaveCCCHandledFlag
Slave received Common command code is handled by I3C module

enumerator kI3C_SlaveEventSentFlag
Slave IBI/P2P/MR/HJ event has been sent

enumerator kI3C_SlaveIbiDisableFlag
Slave in band interrupt is disabled.

enumerator kI3C_SlaveMasterRequestDisabledFlag
Slave master request is disabled.

enumerator kI3C_SlaveHotJoinDisabledFlag
Slave Hot-Join is disabled.

enumerator kI3C_SlaveClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kI3C_SlaveAllIrqFlags

enum _i3c_slave_error_flags
I3C slave error flags to indicate the causes.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI3C_SlaveErrorOverrunFlag
Slave internal from-bus buffer/FIFO overrun.

enumerator kI3C_SlaveErrorUnderrunFlag
Slave internal to-bus buffer/FIFO underrun

enumerator kI3C_SlaveErrorUnderrunNakFlag
Slave internal from-bus buffer/FIFO underrun and NACK error

enumerator kI3C_SlaveErrorTermFlag
Terminate error from master

enumerator kI3C_SlaveErrorInvalidStartFlag
Slave invalid start flag

enumerator kI3C_SlaveErrorSdrParityFlag
SDR parity error

enumerator kI3C_SlaveErrorHdrParityFlag
HDR parity error

enumerator kI3C_SlaveErrorHdrCRCFlag
HDR-DDR CRC error

enumerator kI3C_SlaveErrorS0S1Flag
S0 or S1 error

enumerator kI3C_SlaveErrorOverreadFlag
Over-read error

enumerator kI3C_SlaveErrorOverwriteFlag
Over-write error

enum _i3c_slave_event
I3C slave.event.

Values:

enumerator kI3C_SlaveEventNormal
Normal mode.

enumerator kI3C_SlaveEventIBI
In band interrupt event.

enumerator kI3C_SlaveEventMasterReq
Master request event.

enumerator kI3C_SlaveEventHotJoinReq
Hot-join event.

enum _i3c_slave_activity_state
I3C slave.activity state.

Values:

enumerator kI3C_SlaveNoLatency
Normal bus operation

enumerator kI3C_SlaveLatency1Ms
1ms of latency.

enumerator kI3C_SlaveLatency100Ms
100ms of latency.

enumerator kI3C_SlaveLatency10S
10s latency.

enum _i3c_slave_transfer_event
Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to *I3C_SlaveTransferNonBlocking()* in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kI3C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kI3C_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kI3C_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI3C_SlaveRequiredTransmitEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI3C_SlaveStartEvent

A start/repeated start was detected.

enumerator kI3C_SlaveHDRCommandMatchEvent

Slave Match HDR Command.

enumerator kI3C_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kI3C_SlaveRequestSentEvent

Slave request event sent.

enumerator kI3C_SlaveReceivedCCCEvent

Slave received CCC event, need to handle by application.

enumerator kI3C_SlaveAllEvents

Bit mask of all available events.

typedef enum *_i3c_slave_event* i3c_slave_event_t

I3C slave.event.

typedef enum *_i3c_slave_activity_state* i3c_slave_activity_state_t

I3C slave.activity state.

typedef struct *_i3c_slave_config* i3c_slave_config_t

Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_i3c_slave_transfer_event* i3c_slave_transfer_event_t

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to *I3C_SlaveTransferNonBlocking()* in order to specify

which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

---

typedef struct *_i3c_slave_transfer* i3c_slave_transfer_t

I3C slave transfer structure.

typedef struct *_i3c_slave_handle* i3c_slave_handle_t

typedef void (*i3c_slave_transfer_callback_t)(I3C_Type *base, *i3c_slave_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I3C_SlaveSetCallback() function after you have created a handle.

Param base
Base address for the I3C instance on which the event occurred.

Param transfer
Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData
Arbitrary pointer-sized value passed from the application.

typedef void (*i3c_slave_isr_t)(I3C_Type *base, void *handle)

Typedef for slave interrupt handler.

struct _i3c_slave_config

#include <fsl_i3c.h> Structure with settings to initialize the I3C slave module.

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Whether to enable slave.

uint8_t staticAddr

Static address.

uint16_t vendorID

Device vendor ID(manufacture ID).

uint32_t partNumber

Device part number info

uint8_t dcr
Device characteristics register information.

uint8_t bcr
Bus characteristics register information.

uint8_t hdrMode
Support hdr mode, could be OR logic in enumeration:i3c_hdr_mode_t.

bool nakAllRequest
Whether to reply NAK to all requests except broadcast CCC.

bool ignoreS0S1Error
Whether to ignore S0/S1 error in SDR mode.

bool offline
Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.

bool matchSlaveStartStop
Whether to assert start/stop status only the time slave is addressed.

uint32_t maxWriteLength
Maximum write length.

uint32_t maxReadLength
Maximum read length.

struct __i3c_slave_transfer
#include <fsl_i3c.h> I3C slave transfer structure.

Public Members

uint32_t event
Reason the callback is being invoked.

uint8_t *txData
Transfer buffer

size_t txDataSize
Transfer size

uint8_t *rxData
Transfer buffer

size_t rxDataSize
Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for *kI3C_SlaveCompletionEvent*.

size_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct _i3c_slave_handle

#include <fsl_i3c.h> I3C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

i3c_slave_transfer_t transfer

I3C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32_t eventMask

Mask of enabled events.

uint32_t transferredCount

Count of bytes transferred.

i3c_slave_transfer_callback_t callback

Callback function called at transfer event.

void *userData

Callback parameter passed to callback.

size_t txFifoSize

Tx Fifo size

2.37 I3C Slave DMA Driver

void I3C_SlaveTransferCreateHandleEDMA(I3C_Type *base, *i3c_slave_edma_handle_t* *handle, *i3c_slave_edma_callback_t* callback, void *userData, *edma_handle_t* *rxDmaHandle, *edma_handle_t* *txDmaHandle)

Create a new handle for the I3C slave DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I3C_SlaveTransferAbortDMA() API shall be called.

For devices where the I3C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.
- rxDmaHandle – Handle for the DMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the DMA transmit channel. Created by the user prior to calling this function.

status_t I3C_SlaveTransferEDMA(I3C_Type *base, *i3c_slave_edma_handle_t* *handle, *i3c_slave_edma_transfer_t* *transfer, uint32_t eventMask)

Prepares for a non-blocking DMA-based transaction on the I3C bus.

The API will do DMA configuration according to the input transfer descriptor, and the data will be transferred when there's bus master requesting transfer from/to this slave. So the timing of call to this API need be aligned with master application to ensure the transfer is executed as expected. Callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The I3C peripheral base address.
- handle – Pointer to the I3C slave driver handle.
- transfer – The pointer to the transfer descriptor.
- eventMask – Bit mask formed by OR'ing together *i3c_slave_transfer_event_t* enumerators to specify which events to send to the callback. The transmit and receive events is not allowed to be enabled.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I3C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.
- kStatus_Fail – The transaction can't be set.

void I3C_SlaveTransferAbortEDMA(I3C_Type *base, *i3c_slave_edma_handle_t* *handle)

Abort a slave edma non-blocking transfer in a early time.

Parameters

- base – I3C peripheral base address
- handle – pointer to i3c_slave_edma_handle_t structure

void I3C_SlaveTransferEDMAHandleIRQ(I3C_Type *base, void *i3cHandle)

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- base – The I3C peripheral base address.
- i3cHandle – Pointer to the I3C slave DMA driver handle.

typedef struct *_i3c_slave_edma_handle* i3c_slave_edma_handle_t

typedef struct *_i3c_slave_edma_transfer* i3c_slave_edma_transfer_t

I3C slave transfer structure.

typedef void (*i3c_slave_edma_callback_t)(I3C_Type *base, *i3c_slave_edma_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave DMA transfer API.

Param base
Base address for the I3C instance on which the event occurred.

Param handle
Pointer to slave DMA transfer handle.

Param transfer
Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData
Arbitrary pointer-sized value passed from the application.

struct __i3c_slave_edma_transfer

#include <fsl_i3c_edma.h> I3C slave transfer structure.

Public Members

uint32_t event

Reason the callback is being invoked.

uint8_t *txData

Transfer buffer

size_t txDataSize

Transfer size

uint8_t *rxData

Transfer buffer

size_t rxDataSize

Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for *kI3C_SlaveCompletionEvent*.

struct _i3c_slave_edma_handle

#include <fsl_i3c_edma.h> I3C slave edma handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

I3C_Type *base

I3C base pointer.

i3c_slave_edma_transfer_t transfer

I3C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

bool isDdrMode

Whether this is HDR-DDR transfer.

uint32_t eventMask

Mask of enabled events.

i3c_slave_edma_callback_t callback

Callback function called at transfer event.

edma_handle_t *rxDmaHandle

Handle for receive DMA channel.

edma_handle_t *txDmaHandle

Handle for transmit DMA channel.

void *userData

Callback parameter passed to callback.

2.38 INPUTMUX: Input Multiplexing Driver

enum _inputmux_connection_t

INPUTMUX connections type.

Values:

enumerator kINPUTMUX_GpioPort8Pin0ToPintsel

enumerator kINPUTMUX_GpioPort8Pin1ToPintsel

enumerator kINPUTMUX_GpioPort8Pin2ToPintsel

enumerator kINPUTMUX_GpioPort8Pin3ToPintsel

enumerator kINPUTMUX_GpioPort8Pin4ToPintsel

enumerator kINPUTMUX_GpioPort8Pin5ToPintsel

enumerator kINPUTMUX_GpioPort8Pin6ToPintsel

enumerator kINPUTMUX_GpioPort8Pin7ToPintsel

enumerator kINPUTMUX_GpioPort8Pin8ToPintsel

enumerator kINPUTMUX_GpioPort8Pin9ToPintsel

enumerator kINPUTMUX_GpioPort8Pin10ToPintsel

enumerator kINPUTMUX_GpioPort8Pin11ToPintsel

enumerator kINPUTMUX_GpioPort8Pin12ToPintsel

enumerator kINPUTMUX_GpioPort8Pin13ToPintsel

enumerator kINPUTMUX_GpioPort8Pin14ToPintsel

enumerator kINPUTMUX_GpioPort8Pin15ToPintsel

enumerator kINPUTMUX_GpioPort8Pin16ToPintsel

enumerator kINPUTMUX_GpioPort8Pin17ToPintsel

enumerator kINPUTMUX_GpioPort8Pin18ToPintsel

enumerator kINPUTMUX_GpioPort8Pin19ToPintsel

enumerator kINPUTMUX_GpioPort8Pin20ToPintsel

enumerator kINPUTMUX_GpioPort8Pin21ToPintsel

enumerator kINPUTMUX_GpioPort8Pin22ToPintsel

enumerator kINPUTMUX_GpioPort8Pin23ToPintsel

enumerator kINPUTMUX_GpioPort8Pin24ToPintsel

enumerator kINPUTMUX_GpioPort8Pin25ToPintsel

enumerator kINPUTMUX_GpioPort8Pin26ToPintsel

enumerator kINPUTMUX_GpioPort8Pin27ToPintsel

enumerator kINPUTMUX_GpioPort8Pin28ToPintsel

enumerator kINPUTMUX_GpioPort8Pin29ToPintsel

enumerator kINPUTMUX_GpioPort8Pin30ToPintsel

enumerator kINPUTMUX_GpioPort8Pin31ToPintsel

enumerator kINPUTMUX_GpioPort10Pin0ToPintsel

enumerator kINPUTMUX_GpioPort10Pin1ToPintsel

enumerator kINPUTMUX_GpioPort10Pin2ToPintsel

enumerator kINPUTMUX_GpioPort10Pin3ToPintsel

enumerator kINPUTMUX_GpioPort10Pin4ToPintsel

enumerator kINPUTMUX_GpioPort10Pin5ToPintsel

enumerator kINPUTMUX_GpioPort10Pin6ToPintsel

enumerator kINPUTMUX_GpioPort10Pin7ToPintsel

enumerator kINPUTMUX_GpioPort10Pin8ToPintsel

enumerator kINPUTMUX_GpioPort10Pin9ToPintsel

enumerator kINPUTMUX_GpioPort10Pin10ToPintsel

enumerator kINPUTMUX_GpioPort10Pin11ToPintsel

enumerator kINPUTMUX_GpioPort10Pin12ToPintsel

enumerator kINPUTMUX_GpioPort10Pin13ToPintsel

enumerator kINPUTMUX_GpioPort10Pin14ToPintsel

enumerator kINPUTMUX_GpioPort10Pin15ToPintsel

enumerator kINPUTMUX_GpioPort10Pin16ToPintsel

enumerator kINPUTMUX_GpioPort10Pin17ToPintsel

DSP Interrupt.

enumerator kINPUTMUX_Flexcomm17ToDspInterrupt

enumerator kINPUTMUX_Flexcomm18ToDspInterrupt

enumerator kINPUTMUX_Flexcomm19ToDspInterrupt

enumerator kINPUTMUX_Flexcomm20ToDspInterrupt

enumerator kINPUTMUX_Pmc1ToDspInterrupt

enumerator kINPUTMUX_Gpio8Irq0ToDspInterrupt

enumerator kINPUTMUX_Gpio8Irq1ToDspInterrupt

enumerator kINPUTMUX_Gpio9Irq0ToDspInterrupt

enumerator kINPUTMUX_Gpio9Irq1ToDspInterrupt

enumerator kINPUTMUX_Gpio10Irq0ToDspInterrupt

enumerator kINPUTMUX_Gpio10Irq1ToDspInterrupt

enumerator kINPUTMUX_Wdt2ToDspInterrupt

enumerator kINPUTMUX_Wdt3ToDspInterrupt

enumerator kINPUTMUX_Mu0BToDspInterrupt

enumerator kINPUTMUX_Mu3BToDspInterrupt

enumerator kINPUTMUX_Utick1ToDspInterrupt

enumerator kINPUTMUX_Mrt1ToDspInterrupt

enumerator kINPUTMUX_OsEventTimerToDspInterrupt

enumerator kINPUTMUX_Ctimer5ToDspInterrupt

enumerator kINPUTMUX_Ctimer6ToDspInterrupt

enumerator kINPUTMUX_Ctimer7ToDspInterrupt

enumerator kINPUTMUX_Rtc1AlarmToDspInterrupt

enumerator kINPUTMUX_Rtc1WakeupToDspInterrupt

enumerator kINPUTMUX_I3c2ToDspInterrupt

enumerator kINPUTMUX_I3c3ToDspInterrupt

enumerator kINPUTMUX_MicfilToDspInterrupt

enumerator kINPUTMUX_HwvadToDspInterrupt

enumerator kINPUTMUX_Dma2Irq0ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq1ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq2ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq3ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq4ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq5ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq6ToDspInterrupt

enumerator kINPUTMUX_Dma2Irq7ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq0ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq1ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq2ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq3ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq4ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq5ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq6ToDspInterrupt

enumerator kINPUTMUX_Dma3Irq7ToDspInterrupt

enumerator kINPUTMUX_GpioInt0ToDspInterrupt

enumerator kINPUTMUX_GpioInt1ToDspInterrupt

enumerator kINPUTMUX_GpioInt2ToDspInterrupt

enumerator kINPUTMUX_GpioInt3ToDspInterrupt

enumerator kINPUTMUX_Sai3ToDspInterrupt

enumerator kINPUTMUX_CtInp0ToFlexcomm17

enumerator kINPUTMUX_CtInp1ToFlexcomm17

enumerator kINPUTMUX_CtInp2ToFlexcomm17

enumerator kINPUTMUX_CtInp3ToFlexcomm17

enumerator kINPUTMUX_Ct5Mat3ToFlexcomm17

enumerator kINPUTMUX_Ct6Mat3ToFlexcomm17

enumerator kINPUTMUX_Ct7Mat3ToFlexcomm17

enumerator kINPUTMUX_GpioInt1BMatchToFlexcomm17

enumerator kINPUTMUX_Cmp0OutToFlexcomm17

enumerator kINPUTMUX_Rtc1IrqToFlexcomm17

enumerator kINPUTMUX_CtInp0ToFlexcomm18

enumerator kINPUTMUX_CtInp1ToFlexcomm18

enumerator kINPUTMUX_CtInp2ToFlexcomm18

enumerator kINPUTMUX_CtInp3ToFlexcomm18

enumerator kINPUTMUX_Ct5Mat3ToFlexcomm18

enumerator kINPUTMUX_Ct6Mat3ToFlexcomm18

enumerator kINPUTMUX_Ct7Mat3ToFlexcomm18

enumerator kINPUTMUX_GpioInt1BMatchToFlexcomm18

enumerator kINPUTMUX_Cmp0OutToFlexcomm18

enumerator kINPUTMUX_Rtc1IrqToFlexcomm18

enumerator kINPUTMUX_CtInp0ToFlexcomm19

enumerator kINPUTMUX_CtInp1ToFlexcomm19

enumerator kINPUTMUX_CtInp2ToFlexcomm19

enumerator kINPUTMUX_CtInp3ToFlexcomm19

enumerator kINPUTMUX_Ct5Mat3ToFlexcomm19

enumerator kINPUTMUX_Ct6Mat3ToFlexcomm19

enumerator kINPUTMUX_Ct7Mat3ToFlexcomm19

enumerator kINPUTMUX_GpioInt1BMatchToFlexcomm19

enumerator kINPUTMUX_Cmp0OutToFlexcomm19

enumerator kINPUTMUX_Rtc1IrqToFlexcomm19

enumerator kINPUTMUX_CtInp0ToFlexcomm20

enumerator kINPUTMUX_CtInp1ToFlexcomm20

enumerator kINPUTMUX_CtInp2ToFlexcomm20

enumerator kINPUTMUX_CtInp3ToFlexcomm20

enumerator kINPUTMUX_Ct5Mat3ToFlexcomm20

enumerator kINPUTMUX_Ct6Mat3ToFlexcomm20

enumerator kINPUTMUX_Ct7Mat3ToFlexcomm20

enumerator kINPUTMUX_GpioInt1BMatchToFlexcomm20

enumerator kINPUTMUX_Cmp0OutToFlexcomm20

enumerator kINPUTMUX_Rtc1IrqToFlexcomm20

enumerator kINPUTMUX_Gpio0Irq0ToAdc0

enumerator kINPUTMUX_Gpio1Irq0ToAdc0

enumerator kINPUTMUX_Gpio2Irq0ToAdc0

enumerator kINPUTMUX_Gpio3Irq0ToAdc0

enumerator kINPUTMUX_Gpio4Irq0ToAdc0

enumerator kINPUTMUX_Gpio5Irq0ToAdc0

enumerator kINPUTMUX_Gpio6Irq0ToAdc0

enumerator kINPUTMUX_Gpio7Irq0ToAdc0

enumerator kINPUTMUX_Gpio8Irq0ToAdc0

enumerator kINPUTMUX_Gpio9Irq0ToAdc0

enumerator kINPUTMUX_Gpio10Irq0ToAdc0

enumerator kINPUTMUX_Sct0Out4ToAdc0

enumerator kINPUTMUX_Sct0Out5ToAdc0

enumerator kINPUTMUX_Sct0Out9ToAdc0

enumerator kINPUTMUX_Ct0Mat3ToAdc0

enumerator kINPUTMUX_Ct1Mat3ToAdc0

enumerator kINPUTMUX_Ct2Mat3ToAdc0

enumerator kINPUTMUX_Ct3Mat3ToAdc0

enumerator kINPUTMUX_Ct4Mat3ToAdc0

enumerator kINPUTMUX_Ct5Mat3ToAdc0

enumerator kINPUTMUX_Ct6Mat3ToAdc0

enumerator kINPUTMUX_Ct7Mat3ToAdc0

enumerator kINPUTMUX_Cmp0OutToAdc0

enumerator kINPUTMUX_Cpu0TxevToAdc0

enumerator kINPUTMUX_Cpu1TxevToAdc0

enumerator kINPUTMUX_EzhvOutToAdc0

enumerator kINPUTMUX_GpioInt0BmatchToAdc0

enumerator kINPUTMUX_GpioInt1BmatchToAdc0

CTmier5 capture input mux.

enumerator kINPUTMUX_CtInp0ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp1ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp2ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp3ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp4ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp5ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp6ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp7ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp8ToTimer5CaptureChannels

enumerator kINPUTMUX_CtInp9ToTimer5CaptureChannels

enumerator kINPUTMUX_Sai3TxSyncToTimer5CaptureChannels

enumerator kINPUTMUX_Sai3RxSyncToTimer5CaptureChannels

enumerator kINPUTMUX_Cmp0OutToTimer5CaptureChannels

enumerator kINPUTMUX_Adc0Tcomp0ToTimer5CaptureChannels

enumerator kINPUTMUX_Adc0Tcomp1ToTimer5CaptureChannels

CTmier6 capture input mux.

enumerator kINPUTMUX_CtInp0ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp1ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp2ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp3ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp4ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp5ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp6ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp7ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp8ToTimer6CaptureChannels

enumerator kINPUTMUX_CtInp9ToTimer6CaptureChannels

enumerator kINPUTMUX_Sai3TxSyncToTimer6CaptureChannels

enumerator kINPUTMUX_Sai3RxSyncToTimer6CaptureChannels

enumerator kINPUTMUX_Cmp0OutToTimer6CaptureChannels

enumerator kINPUTMUX_Adc0Tcomp0ToTimer6CaptureChannels

enumerator kINPUTMUX_Adc0Tcomp1ToTimer6CaptureChannels
CTmier7 capture input mux.

enumerator kINPUTMUX_CtInp0ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp1ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp2ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp3ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp4ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp5ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp6ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp7ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp8ToTimer7CaptureChannels

enumerator kINPUTMUX_CtInp9ToTimer7CaptureChannels

enumerator kINPUTMUX_Sai3TxSyncToTimer7CaptureChannels

enumerator kINPUTMUX_Sai3RxSyncToTimer7CaptureChannels

enumerator kINPUTMUX_Cmp0OutToTimer7CaptureChannels

enumerator kINPUTMUX_Adc0Tcomp0ToTimer7CaptureChannels

enumerator kINPUTMUX_Adc0Tcomp1ToTimer7CaptureChannels
CTmier5 input trigger.

enumerator kINPUTMUX_CtInp0ToTimer5Trigger

enumerator kINPUTMUX_CtInp1ToTimer5Trigger

enumerator kINPUTMUX_CtInp2ToTimer5Trigger

enumerator kINPUTMUX_CtInp3ToTimer5Trigger

enumerator kINPUTMUX_CtInp4ToTimer5Trigger

enumerator kINPUTMUX_CtInp5ToTimer5Trigger

enumerator kINPUTMUX_CtInp6ToTimer5Trigger

enumerator kINPUTMUX_CtInp7ToTimer5Trigger

enumerator kINPUTMUX_CtInp8ToTimer5Trigger

enumerator kINPUTMUX_CtInp9ToTimer5Trigger

enumerator kINPUTMUX_Sai3TxSyncToTimer5Trigger

enumerator kINPUTMUX_Sai3RxSyncToTimer5Trigger

enumerator kINPUTMUX_Cmp0OutToTimer5Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToTimer5Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToTimer5Trigger
CTmier6 input trigger.

enumerator kINPUTMUX_CtInp0ToTimer6Trigger

enumerator kINPUTMUX_CtInp1ToTimer6Trigger

enumerator kINPUTMUX_CtInp2ToTimer6Trigger

enumerator kINPUTMUX_CtInp3ToTimer6Trigger

enumerator kINPUTMUX_CtInp4ToTimer6Trigger

enumerator kINPUTMUX_CtInp5ToTimer6Trigger

enumerator kINPUTMUX_CtInp6ToTimer6Trigger

enumerator kINPUTMUX_CtInp7ToTimer6Trigger

enumerator kINPUTMUX_CtInp8ToTimer6Trigger

enumerator kINPUTMUX_CtInp9ToTimer6Trigger

enumerator kINPUTMUX_Sai3TxSyncToTimer6Trigger

enumerator kINPUTMUX_Sai3RxSyncToTimer6Trigger

enumerator kINPUTMUX_Cmp0OutToTimer6Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToTimer6Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToTimer6Trigger

CTmier7 input trigger.

enumerator kINPUTMUX_CtInp0ToTimer7Trigger

enumerator kINPUTMUX_CtInp1ToTimer7Trigger

enumerator kINPUTMUX_CtInp2ToTimer7Trigger

enumerator kINPUTMUX_CtInp3ToTimer7Trigger

enumerator kINPUTMUX_CtInp4ToTimer7Trigger

enumerator kINPUTMUX_CtInp5ToTimer7Trigger

enumerator kINPUTMUX_CtInp6ToTimer7Trigger

enumerator kINPUTMUX_CtInp7ToTimer7Trigger

enumerator kINPUTMUX_CtInp8ToTimer7Trigger

enumerator kINPUTMUX_CtInp9ToTimer7Trigger

enumerator kINPUTMUX_Sai3TxSyncToTimer7Trigger

enumerator kINPUTMUX_Sai3RxSyncToTimer7Trigger

enumerator kINPUTMUX_Cmp0OutToTimer7Trigger

enumerator kINPUTMUX_Adc0Tcomp0ToTimer7Trigger

enumerator kINPUTMUX_Adc0Tcomp1ToTimer7Trigger

typedef enum *_inputmux_connection_t* inputmux_connection_t

INPUTMUX connections type.

PINTSEL_PMUX_ID

Periphinmux IDs.

DSP_INT_PMUX_ID

PMUX_SHIFT

FSL_INPUTMUX_DRIVER_VERSION

Group interrupt driver version for SDK.

void INPUTMUX_Init(void *base)

Initialize INPUTMUX peripheral.

This function enables the INPUTMUX clock.

Parameters

- base – Base address of the INPUTMUX peripheral.

void INPUTMUX_AttachSignal(void *base, uint16_t index, *inputmux_connection_t* connection)

Attaches a signal.

This function writes a source signal selection into an INPUTMUX multiplexer register. The target register address is computed as:

```
target address = base + pmux_id + index * 4
```

Each *inputmux_connection_t* enum value encodes two fields via PMUX_SHIFT (20):

```
31          20 19                    0
+------------+------------------------+
| pmux_id    |       output_id        |
+------------+------------------------+
     |                 |
group base          signal value
addr offset         to write
```

When N consecutive registers all accept the same set of source signals (same pmux_id), only one set of enum values is defined for the whole group; index (0 ~ N-1) selects which register to write, avoiding duplicate enum entries.

Example: SCT0 has 8 input mux registers sharing SCT0_INMUX0 = 0x000 as the group base:

```
base
 |
 +--[+0x000] SCT0_INMUX0  <-- index=0
 +--[+0x004] SCT0_INMUX1  <-- index=1
 +--[+0x008] SCT0_INMUX2  <-- index=2
 :   ...
 +--[+0x014] SCT0_INMUX5  <-- index=5  (write target for the call below)
 :   ...
 +--[+0x01C] SCT0_INMUX7  <-- index=7
```

To connect CMP0 output to SCT0 input 5:

```
INPUTMUX_AttachSignal(INPUTMUX, 5, kINPUTMUX_Cmp0OutToSct0);
// write address = base + 0x000 + 5*4 = base + 0x014  (SCT0_INMUX5)
```

For peripherals with only a single register per function (e.g. FREQMEAS_REF_REG = 0x180, FREQMEAS_TAR_REG = 0x184), each register has its own unique pmux_id and index must be 0. The two registers are distinguished by their different pmux_id values, not by index:

```
base
 |
 +--[+0x180] FREQMEAS_REF_REG  (pmux_id=0x180, index=0)
 +--[+0x184] FREQMEAS_TAR_REG  (pmux_id=0x184, index=0)
```

Parameters

- base – Base address of the INPUTMUX peripheral.
- index – Zero-based index of the destination register within its group. Each increment advances the address by 4 bytes. Use 0 for single-register groups.
- connection – Encodes the group base offset (bits [31:PMUX_SHIFT]) and the source signal value (bits [PMUX_SHIFT-1:0]).

void INPUTMUX_Deinit(void *base)

Deinitialize INPUTMUX peripheral.

This function disables the INPUTMUX clock.

Parameters

- base – Base address of the INPUTMUX peripheral.

2.39 Iopctl_driver

typedef struct *_iopctl_group* iopctl_group_t

Array of IOPCTL pin definitions passed to *IOPCTL_SetPinMuxing()* must be in this format.

__STATIC_INLINE void IOPCTL_PinMuxSet (uint8_t port, uint8_t pin, uint32_t modefunc)

Sets I/O Pad Control pin mux.

Parameters

- port – : Port to mux
- pin – : Pin to mux
- modefunc – : OR'ed values of type IOPCTL_*

Returns
Nothing

___STATIC_INLINE void IOPCTL_SetPinMuxing (const iopctl_group_t *pinArray, uint32_t arrayLength)

Set all I/O Control pin muxing.

Parameters

- pinArray – : Pointer to array of pin mux selections
- arrayLength – : Number of entries in pinArray

Returns
Nothing

LPC_IOPCTL_DRIVER_VERSION

IOPCTL driver version 2.0.3.

IOPCTL_FUNC0

IOPCTL function and mode selection definitions.

Note: See the User Manual for specific modes and functions supported by the various pins.
Selects pin function 0

IOPCTL_FUNC1

Selects pin function 1

IOPCTL_FUNC2

Selects pin function 2

IOPCTL_FUNC3

Selects pin function 3

IOPCTL_FUNC4

Selects pin function 4

IOPCTL_FUNC5

Selects pin function 5

IOPCTL_FUNC6

Selects pin function 6

IOPCTL_FUNC7

Selects pin function 7

IOPCTL_FUNC8

Selects pin function 8

IOPCTL_FUNC9
: Selects pin function 9

IOPCTL_FUNC10
: Selects pin function 10

IOPCTL_FUNC11
: Selects pin function 11

IOPCTL_FUNC12
: Selects pin function 12

IOPCTL_FUNC13
: Selects pin function 13

IOPCTL_FUNC14
: Selects pin function 14

IOPCTL_FUNC15
: Selects pin function 15

IOPCTL_PUPD_EN
: Enables Pullup / Pulldown

IOPCTL_PULLDOWN_EN
: Selects pull-down function

IOPCTL_PULLUP_EN
: Selects pull-up function

IOPCTL_INBUF_EN
: Enables buffer function on input

IOPCTL_SLEW_RATE
: Slew Rate Control

IOPCTL_ANAMUX_EN
: Enables analog mux function by setting 0 to bit 7

IOPCTL_PSEDRAIN_EN
: Enables pseudo output drain function

IOPCTL_INV_EN
: Enables invert function on input

IOPCTL_DRIVE_100OHM
: Selects transmitter current drive 100ohm, avaiable for PIO0,1,3

IOPCTL_DRIVE_66OHM
: Selects transmitter current drive 66ohm, avaiable for PIO0,1,3

IOPCTL_DRIVE_50OHM
: Selects transmitter current drive 50ohm, avaiable for PIO0,1,3

IOPCTL_DRIVE_33OHM
: Selects transmitter current drive 33ohm, avaiable for PIO0,1,3

IOPCTL_FULLDRIVE_EN
: Selects full drive, avaiable for PIO2,8,9,10. (Unsupported for A0 Silicon)

uint32_t port

uint32_t pin

uint32_t modefunc

FSL_COMPONENT_ID

struct _iopctl_group
: *#include <fsl_iopctl.h>* Array of IOPCTL pin definitions passed to *IOPCTL_SetPinMuxing()* must be in this format.

2.40 IRTC: IRTC Driver

status_t IRTC_Init(RTC_Type *base, const *irtc_config_t* *config)
: Ungates the IRTC clock and configures the peripheral for basic operation.

  This function initiates a soft-reset of the IRTC module, this has not effect on DST, calendaring, standby time and tamper detect registers.

  Note: This API should be called at the beginning of the application using the IRTC driver.

  Parameters

  - base – IRTC peripheral base address
  - config – Pointer to user's IRTC config structure.

  Returns
  kStatus_Success If the driver is initialized successfully.

  Returns
  kStatus_Fail if we cannot disable register write protection

  Returns
  kStatus_InvalidArgument If the input parameters are wrong.

status_t IRTC_Deinit(RTC_Type *base)

Gate the IRTC clock.

Parameters

- base – IRTC peripheral base address

Returns

kStatus_Success If the driver is initialized successfully.

Returns

kStatus_InvalidArgument If the input parameters are wrong.

void IRTC_GetDefaultConfig(*irtc_config_t* *config)

Fill in the IRTC config struct with the default settings.

The default values are:

```
config->wakeupSelect = true;
config->timerStdMask = false;
config->alrmMatch = kRTC_MatchSecMinHr;
```

Parameters

- config – Pointer to user's IRTC config structure.

void IRTC_GetDatetime(RTC_Type *base, *irtc_datetime_t* *datetime)

Gets the IRTC time and stores it in the given time structure.

Parameters

- base – IRTC peripheral base address
- datetime – Pointer to structure where the date and time details are stored.

status_t IRTC_SetAlarm(RTC_Type *base, const *irtc_datetime_t* *alarmTime)

Sets the IRTC alarm time.

Note: weekDay field of alarmTime is not used during alarm match and should be set to 0

Parameters

- base – RTC peripheral base address
- alarmTime – Pointer to structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the alarm kStatus_InvalidArgument: error in setting the alarm. Error occurs because the alarm datetime format is incorrect.

void IRTC_GetAlarm(RTC_Type *base, *irtc_datetime_t* *datetime)

Returns the IRTC alarm time.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to structure where the alarm date and time details are stored.

static inline void IRTC_EnableInterrupts(RTC_Type *base, uint32_t mask)

Enables the selected IRTC interrupts.

Parameters

- base – IRTC peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *irtc_interrupt_enable_t*

static inline void IRTC_DisableInterrupts(RTC_Type *base, uint32_t mask)

Disables the selected IRTC interrupts.

Parameters

- base – IRTC peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *irtc_interrupt_enable_t*

static inline uint32_t IRTC_GetEnabledInterrupts(RTC_Type *base)

Gets the enabled IRTC interrupts.

Parameters

- base – IRTC peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *irtc_interrupt_enable_t*

static inline uint32_t IRTC_GetStatusFlags(RTC_Type *base)

Gets the IRTC status flags.

Parameters

- base – IRTC peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration *irtc_status_flags_t*

static inline void IRTC_ClearStatusFlags(RTC_Type *base, uint32_t mask)

Clears the IRTC status flags.

Parameters

- base – IRTC peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration *irtc_status_flags_t*

void IRTC_GetDaylightTime(RTC_Type *base, *irtc_daylight_time_t* *datetime)

Gets the IRTC daylight savings time and stores it in the given time structure.

Parameters

- base – IRTC peripheral base address
- datetime – Pointer to a structure where the date and time details are stored.

static inline void IRTC_EnableSubsecondCounter(RTC_Type *base, bool enable)

Enable the RTC wake-up timer.

1HZ clock out selected via call to API IRTC_ConfigClockOut in order for the subsecond counter to synchronize with the RTC_SECONDS counter.

Parameters

- base – RTC peripheral base address

- enable – Use/Un-use the sub-second counter.
  - true: Use RTC wake-up timer at the same time.
  - false: Un-use RTC wake-up timer, RTC only use the normal seconds timer by default.

static inline uint32_t IRTC_GetSubsecondCount(RTC_Type *base)

Read the actual RTC sub-second COUNT value.

Parameters

- base – RTC peripheral base address

Returns

The actual RTC sub-second COUNT value.

static inline void IRTC_SetWakeupCount(RTC_Type *base, bool enable1kHzClk, uint32_t wakeupValue)

Set countdown value to the RTC wake timer counter register.

Parameters

- base – RTC peripheral base address
- enable1kHzClk – Enable 1kHz clock source for the wake timer, else use the 32kHz clock.
- wakeupValue – The value to be loaded into the WAKE register in wake timer counter.

static inline uint32_t IRTC_GetWakeupCount(RTC_Type *base)

Read the actual value from the WAKE register value in RTC wake timer.

Parameters

- base – RTC peripheral base address

Returns

The actual value of the WAKE register value in wake timer counter.

FSL_IRTC_DRIVER_VERSION

enum _irtc_clock_select

IRTC clock select.

Values:

enumerator kIRTC_Clk16K

16.384 kHz clock is selected.

enumerator kIRTC_Clk32K

32.768 kHz clock is selected.

enum _irtc_interrupt_enable

List of IRTC interrupts.

Values:

enumerator kIRTC_AlarmInterruptEnable

Alarm Interrupt Enable

enumerator kIRTC_DayInterruptEnable
Days Interrupt Enable

enumerator kIRTC_HourInterruptEnable
Hours Interrupt Enable

enumerator kIRTC_MinInterruptEnable
Minutes Interrupt Enable

enumerator kIRTC_1hzInterruptEnable
1 Hz interval Interrupt Enable

enumerator kIRTC_2hzInterruptEnable
2 Hz interval Interrupt Enable

enumerator kIRTC_4hzInterruptEnable
4 Hz interval Interrupt Enable

enumerator kIRTC_8hzInterruptEnable
8 Hz interval Interrupt Enable

enumerator kIRTC_16hzInterruptEnable
16 Hz interval Interrupt Enable

enumerator kIRTC_32hzInterruptEnable
32 Hz interval Interrupt Enable

enumerator kIRTC_64hzInterruptEnable
64 Hz interval Interrupt Enable

enumerator kIRTC_128hzInterruptEnable
128 Hz interval Interrupt Enable

enumerator kIRTC_256hzInterruptEnable
256 Hz interval Interrupt Enable

enumerator kIRTC_512hzInterruptEnable
512 Hz interval Interrupt Enable

enumerator kIRTC_WakeTimerInterruptEnable
Wake timer Interrupt Enable

enumerator kIRTC_TamperQueueFullInterruptEnable
Tamper queue full Interrupt Enable

enum _irtc_status_flags
List of IRTC flags.
Values:

enumerator kIRTC_AlarmFlag
Alarm Status flag

enumerator kIRTC_DayFlag
Days Status flag

enumerator kIRTC_HourFlag
Hour Status flag

enumerator kIRTC_MinFlag
Minutes Status flag

enumerator kIRTC_1hzFlag
1 Hz interval status flag

enumerator kIRTC_2hzFlag
2 Hz interval status flag

enumerator kIRTC_4hzFlag
4 Hz interval status flag

enumerator kIRTC_8hzFlag
8 Hz interval status flag

enumerator kIRTC_16hzFlag
16 Hz interval status flag

enumerator kIRTC_32hzFlag
32 Hz interval status flag

enumerator kIRTC_64hzFlag
64 Hz interval status flag

enumerator kIRTC_128hzFlag
128 Hz interval status flag

enumerator kIRTC_256hzFlag
256 Hz interval status flag

enumerator kIRTC_512hzFlag
512 Hz interval status flag

enumerator kIRTC_InvalidFlag
Indicates if time/date counters are invalid

enumerator kIRTC_WriteProtFlag
Write protect enable status flag

enumerator kIRTC_BusErrFlag
Bus error flag

enumerator kIRTC_WakeTimerFlag
Wake timer status flag

enum _irtc_alarm_match
IRTC alarm match options.
Values:

enumerator kRTC_MatchSecMinHr
Only match second, minute and hour

enumerator kRTC_MatchSecMinHrDay
Only match second, minute, hour and day

enumerator kRTC_MatchSecMinHrDayMnth
Only match second, minute, hour, day and month

enumerator kRTC_MatchSecMinHrDayMnthYr
Only match second, minute, hour, day, month and year

enum _irtc_clockout_sel
IRTC clockout select.
Values:

enumerator kIRTC_ClkoutNo
No clock out

enumerator kIRTC_ClkoutFine1Hz
clock out fine 1Hz

enumerator kIRTC_Clkout32kHz
clock out 32.768kHz

enumerator kIRTC_ClkoutCoarse1Hz
clock out coarse 1Hz

typedef enum *_irtc_clock_select* irtc_clock_select_t
IRTC clock select.

typedef enum *_irtc_interrupt_enable* irtc_interrupt_enable_t
List of IRTC interrupts.

typedef enum *_irtc_status_flags* irtc_status_flags_t
List of IRTC flags.

typedef enum *_irtc_alarm_match* irtc_alarm_match_t

IRTC alarm match options.

typedef enum *_irtc_clockout_sel* irtc_clockout_sel_t

IRTC clockout select.

typedef struct *_irtc_datetime* irtc_datetime_t

Structure is used to hold the date and time.

typedef struct *_irtc_daylight_time* irtc_daylight_time_t

Structure is used to hold the daylight saving time.

typedef struct *_irtc_config* irtc_config_t

RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the *IRTC_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

status_t IRTC_SetWriteProtection(RTC_Type *base, bool lock)

Locks or unlocks IRTC registers for write access.

Note: When the registers are unlocked, they remain in unlocked state for 2 seconds, after which they are locked automatically. After power-on-reset, the registers come out unlocked and they are locked automatically 15 seconds after power on.

Parameters

- base – IRTC peripheral base address
- lock – true: Lock IRTC registers; false: Unlock IRTC registers.

Returns

kStatus_Success: if lock or unlock operation is successful kStatus_Fail: if lock or unlock operation fails even after multiple retry attempts

static inline void IRTC_Reset(RTC_Type *base)

Performs a software reset on the IRTC module.

Clears contents of alarm, interrupt (status and enable except tamper interrupt enable bit) registers, STATUS[CMP_DONE] and STATUS[BUS_ERR]. This has no effect on DST, calendaring, standby time and tamper detect registers.

Parameters

- base – IRTC peripheral base address

void IRTC_ConfigClockOut(RTC_Type *base, *irtc_clockout_sel_t* clkOut)

Select which clock to output from RTC.

Select which clock to output from RTC for other modules to use inside SoC, for example, RTC subsystem needs RTC to output 1HZ clock for sub-second counter.

Parameters

- base – IRTC peripheral base address
- clkOut – select clock to use for output,

void IRTC_ConfigClockSelect(RTC_Type *base, *irtc_clock_select_t* clkSelect)

Select which clock is used by RTC.

Select which clock is used by RTC to output to the peripheral and divided to generate a 512 Hz clock and a 1 Hz clock.

Parameters

- base – IRTC peripheral base address
- clkSelect – select clock used by RTC

static inline void IRTC_EnableClockOutputToPeripheral(RTC_Type *base, bool enable)

Determines whether the selected clock is output to other peripherals.

Determines whether the selected clock is output to other peripherals.

Parameters

- base – IRTC peripheral base address
- enable – determine whether the selected clock is output to other peripherals

IRTC_STATUS_W1C_BITS

struct _irtc_datetime

#include <fsl_irtc.h> Structure is used to hold the date and time.

Public Members

uint16_t year

Range from 1984 to 2239.

uint8_t month

Range from 1 to 12.

uint8_t day

Range from 1 to 31 (depending on month).

uint8_t weekDay

Range from 0(Sunday) to 6(Saturday).

uint8_t hour

Range from 0 to 23.

uint8_t minute

Range from 0 to 59.

uint8_t second

Range from 0 to 59.

struct _irtc_daylight_time

#include <fsl_irtc.h> Structure is used to hold the daylight saving time.

Public Members

uint8_t startMonth

Range from 1 to 12

uint8_t endMonth

Range from 1 to 12

uint8_t startDay

Range from 1 to 31 (depending on month)

uint8_t endDay

Range from 1 to 31 (depending on month)

uint8_t startHour

Range from 0 to 23

uint8_t endHour

Range from 0 to 23

struct _irtc_config

#include <fsl_irtc.h> RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the *IRTC_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Public Members

irtc_alarm_match_t alrmMatch

Pick one option from enumeration :: irtc_alarm_match_t

irtc_clock_select_t clockSelect

Pick one option from enumeration :: irtc_clock_select_t

bool disableClockOutput

true: The selected clock is not output to other peripherals; false: The selected clock is output to other peripherals

2.41 ITRC

status_t ITRC_SetActionToEvent(ITRC_Type *base, *itrc_out_signals_t* out, *itrc_input_signals_t* in, *itrc_lock_t* lock, *itrc_enable_t* enable)

Set ITRC Action to Event.

This function sets input Event signal to corresponding output Action response signal.

Parameters

- base – ITRC peripheral base address
- out – ITRC OUT signal action
- in – ITRC IN signal event
- lock – if set locks INx_SEL configuration. This can be cleared only by PMC Core reset.
- enable – if set input Event will be selected for output Action, otherwise disable (if not already locked).

Returns
kStatus_Success if success, kStatus_InvalidArgument otherwise

void ITRC_SetSWEvent0(ITRC_Type *base)

Trigger ITRC SW Event 0.

This function set SW_EVENT0 register with value !=0 which triggers ITRC SW Event 0.

Parameters

- base – ITRC peripheral base address

void ITRC_SetSWEvent1(ITRC_Type *base)

Trigger ITRC SW Event 1.

This function set SW_EVENT1 register with value !=0 which triggers ITRC SW Event 1.

Parameters

- base – ITRC peripheral base address

uint32_t ITRC_GetStatus(ITRC_Type *base)

Get ITRC Status.

This function returns ITRC register status.

Parameters

- base – ITRC peripheral base address

Returns
Value of ITRC STATUS register

status_t ITRC_ClearStatus(ITRC_Type *base, uint32_t word)

Clear ITRC status.

This function clears corresponding ITRC event or action in STATUS register.

Parameters

- base – ITRC peripheral base address
- word – 32bit word represent corresponding event/action in STATUS register to be cleared (see ITRC_STATUS_INx/OUTx_STATUS)

Returns
kStatus_Success if success, kStatus_InvalidArgument otherwise

uint32_t ITRC_GetStatus1(ITRC_Type *base)

Get ITRC Status 1.

This function returns ITRC STATUS1 register value.

Parameters

- base – ITRC peripheral base address

Returns
Value of ITRC STATUS1 register, if register STATUS1 doesn't exist it return 0

status_t ITRC_ClearStatus1(ITRC_Type *base, uint32_t word)

brief Clear ITRC status 1

This function clears corresponding ITRC event or action in STATUS1 register.

param base ITRC peripheral base address param word 32bit word represent corresponding event/action in STATUS1 register to be cleared (see ITRC_STATUS_INx/OUTx_STATUS) return kStatus_Success if success, kStatus_InvalidArgument otherwise

status_t ITRC_ClearAllStatus(ITRC_Type *base)

Clear All ITRC status.

This function clears all event and action status.

Parameters

- base – ITRC peripheral base address

Returns

kStatus_Success if success

status_t ITRC_Init(ITRC_Type *base)

Initialize ITRC.

This function initializes ITRC by enabling IRQ.

Parameters

- base – ITRC peripheral base address
- conf – ITRC configuration structure

Returns

Status of the init operation

void ITRC_Deinit(ITRC_Type *base)

Deinitialize ITRC.

This function deinitializes ITRC by disabling IRQ.

Parameters

- base – ITRC peripheral base address

FSL_ITRC_DRIVER_VERSION

Defines ITRC driver version 2.4.2.

Change log:

- Version 2.4.2
  - Update to KW43 ITRC offering
- Version 2.4.1
  - Update to MCXA577 ITRC offering
- Version 2.4.0
  - Rework the input signal definition for better flexibility
- Version 2.3.0
  - Update names of kITRC_SwEvent1/2 to kITRC_SwEvent0/1 to align with RM
- Version 2.2.0
  - Update driver to new version and input events
- Version 2.1.0
  - Make SYSCON glitch platform dependent

- Version 2.0.0
  - initial version

enum _itrc_input_signals

Values:

enum _itrc_lock

Values:

enumerator kITRC_Unlock

enumerator kITRC_Lock

enum _itrc_enable

Values:

enumerator kITRC_Enable

enumerator kITRC_Disable

enum _itrc_out_signals

Values:

enumerator kITRC_Out_0

enumerator kITRC_Out_1

enumerator kITRC_Out_2

enumerator kITRC_Out_3

enumerator kITRC_Out_4

enumerator kITRC_Out_5

enumerator kITRC_Out_6

typedef enum *_itrc_input_signals* itrc_input_signals_t

typedef enum *_itrc_lock* itrc_lock_t

typedef enum *_itrc_enable* itrc_enable_t

typedef enum *_itrc_out_signals* itrc_out_signals_t

void ITRC0_DriverIRQHandler(void)

ITRC_STATUS_IN0_STATUS_MASK

ITRC_STATUS_IN1_STATUS_MASK

ITRC_STATUS_IN2_STATUS_MASK

ITRC_STATUS_IN3_STATUS_MASK

ITRC_STATUS_IN6_STATUS_MASK

ITRC_STATUS_IN7_STATUS_MASK

ITRC_STATUS_IN8_STATUS_MASK

ITRC_STATUS_IN9_STATUS_MASK

ITRC_STATUS_IN14_STATUS_MASK

ITRC_STATUS1_IN17_STATUS_MASK

ITRC_STATUS1_IN19_STATUS_MASK

ITRC_STATUS1_IN24_21_STATUS_MASK

ITRC_STATUS1_IN32_25_STATUS_MASK

ITRC_STATUS1_IN36_STATUS_MASK

ITRC_STATUS1_IN46_STATUS_MASK

IN_0_15_EVENTS_MASK

OUT_ACTIONS_MASK

ITRC_OUT_COUNT

ITRC

2.42 Intrusion and Tamper Response Controller

2.43 JPEGDEC: JPEG decoder Driver

Error codes for the JPEG decoder driver.

Values:

enumerator kStatus_JPEGDEC_NotSupported
Unsupported pixel format or image resolution.

enum _jpegdec_pixel_format
JPEG decoding pixel format.

Values:

enumerator kJPEGDEC_PixelFormatYUV420
YUV420 format, Y at first plane, UV at second plane.

enumerator kJPEGDEC_PixelFormatYUV422
YUV422, 1 planar in the YUYV sequence.

enumerator kJPEGDEC_PixelFormatRGB
RGB packed format.

enumerator kJPEGDEC_PixelFormatYUV444
YUV444, 1 planar in the YUVYUV sequence.

enumerator kJPEGDEC_PixelFormatGray
8-bit or 12-bit gray scale.

enumerator kJPEGDEC_PixelFormatReserved
Reserved.

enumerator kJPEGDEC_PixelFormatYCCK
YCCK format.

enum _jpegdec_pixel_depth
JPEG decoding pixel depth.

Values:

enumerator kJPEGDEC_PixelDepth8Bit
8-bit per pixel.

enumerator kJPEGDEC_PixelDepth12Bit
12-bit per pixel.

enum _jpegdec_flags
Decoder slot Interrupt status and enable/disable flags.

Values:

enumerator kJPEGDEC_StreamBufferHalfDoneFlag
Slot's bit stream buffer pointer has passed half the size.

enumerator kJPEGDEC_StreamBufferDoneFlag
Slot's bit stream buffer pointer has finished.

enumerator kJPEGDEC_SwitchInFlag
Current slot is switched in.

enumerator kJPEGDEC_DecodeCompleteFlag
Current slot finished decoding.

enumerator kJPEGDEC_DecodeErrorFlag
Current slot has error during decoding.

enumerator kJPEGDEC_DescptReadErrorFlag
Descriptor read error when switching in current slot.

enumerator kJPEGDEC_BitReadErrorFlag
Read JPEG stream error.

enumerator kJPEGDEC_PixelWriteErrorFlag
Write decoded pixel error.

enumerator kJPEGDEC_ErrorFlags

enumerator kJPEGDEC_AllFlags

enum _jpegdec_slots
JPEG decoding Slot.
Values:

enumerator kJPEGDEC_Slot0
Decoding slot 0.

enumerator kJPEGDEC_Slot1
Decoding slot 1.

enumerator kJPEGDEC_Slot2
Decoding slot 2.

enumerator kJPEGDEC_Slot3
Decoding slot 3.

enum _jpegdec_endian_mode
JPEG stream data format.
Values:

enumerator kJPEGDEC_BigEndian
JPEG data format is big endian.

enumerator kJPEGDEC_LittleEndian

JPEG data format is little endian.

typedef enum *_jpegdec_pixel_format* jpegdec_pixel_format_t

JPEG decoding pixel format.

typedef enum *_jpegdec_pixel_depth* jpegdec_pixel_depth_t

JPEG decoding pixel depth.

typedef struct *_jpegdec_decoder_config* jpegdec_decoder_config_t

JPEG decoder configuration.

typedef struct *_jpegdec_descpt* jpegdec_descpt_t

JPEG decoder descriptor.

This structure is used to configure the linked decoding descriptor for one or more slots in Single Bit Stream Repeat mode or Context Switch mode.

typedef enum *_jpegdec_endian_mode* jpegdec_endian_mode_t

JPEG stream data format.

typedef struct *_jpegdec_config* jpegdec_config_t

Configuration for JPEG decoder.

FSL_JPEGDEC_DRIVER_VERSION

JPEG decoder driver version.

void JPEGDEC_GetDefaultConfig(*jpegdec_config_t* *config)

Gets the default configuration for JPEG decoder.

This function initializes the user configuration structure to default value. The default value are:

Example:

```
config->endianMode = kJPEGDEC_LittleEndian;
config->slots      = 0xFU;
config->enable     = true;
config->enableLp   = false;
```

Parameters

- config – Pointer to JPEG decoder configuration structure.

void JPEGDEC_Init(*JPEG_DECODER_Type* *base, const *jpegdec_config_t* *config)

Initializes the JPEG decoder.

The default configuration can be got by calling *JPEGDEC_GetDefaultConfig*.

Parameters

- base – JPEG decoder peripheral base address.
- config – Pointer to JPEG decoder configuration structure.

void JPEGDEC_Deinit(*JPEG_DECODER_Type* *base)

Deinitializes the JPEG decoder.

Parameters

- base – JPEG decoder peripheral base address.

static inline void JPEGDEC_EnableInterrupts(*JPEG_DECODER_Type* *base, uint8_t slot, uint32_t mask)

Enables interrupts for certain slot.

Parameters

- base – JPEG decoder peripheral base address.
- slot – Slot number.
- mask – Interrupts mask formed by flags OR'ed together. *_jpegdec_flags*.

static inline void JPEGDEC_DisableInterrupts(*JPEG_DECODER_Type* *base, uint8_t slot, uint32_t mask)

Disables interrupts for certain slot.

Parameters

- base – JPEG decoder peripheral base address.
- slot – Slot number.
- mask – Interrupts mask formed by flags OR'ed together. *_jpegdec_flags*.

static inline uint32_t JPEGDEC_GetStatusFlags(*JPEG_DECODER_Type* *base, uint8_t slot)

Gets status flags for certain slot.

Parameters

- base – JPEG decoder peripheral base address.
- slot – Slot number.

Returns

Status flags asserted mask formed by flags OR'ed together. See "_jpegdec_flags".

static inline void JPEGDEC_ClearStatusFlags(*JPEG_DECODER_Type* *base, uint8_t slot, uint32_t mask)

Clears status flags for certain slot.

Parameters

- base – JPEG decoder peripheral base address.
- slot – Slot number.
- mask – Status flags mask formed by flags OR'ed together. See "_jpegdec_flags".

static inline void JPEGDEC_Enable(*JPEG_DECODER_Type* *base, bool enable)

Enables the JPEG decoder module.

Parameters

- base – JPEG decoder peripheral base address.
- enable – True to enable, false to disable.

static inline void JPEGDEC_EnableLp(*JPEG_DECODER_Type* *base, bool enable)

Enables the JPEG decoder core to enter low power mode.

Parameters

- base – JPEG decoder peripheral base address.
- enable – True to enable, false to disable.

static inline void JPEGDEC_Reset(*JPEG_DECODER_Type* *base)

Resets the JPEG decoder.

Parameters

- base – JPEG decoder peripheral base address.

static inline void JPEGDEC_StartDecode(*JPEG_DECODER_Type* *base)

Starts the JPEG decode.

After calling this API the JPEG decoder will start decoding on a frame basis. The decoding process stops after the frame decoding is complete and if there are no other valid descriptors available. Auto-clear.

Parameters

- base – JPEG decoder peripheral base address.

static inline void JPEGDEC_EnableSlots(*JPEG_DECODER_Type* *base, uint8_t slots)

Enables the JPEG decoding slot(s)

Parameters

- base – JPEG decoder peripheral base address.
- slot – Masks of the enabled slot, *_jpegdec_slots*.

status_t JPEGDEC_GetActiveSlot(*JPEG_DECODER_Type* *base, uint8_t *slot)

Gets the current active slot.

Parameters

- base – JPEG decoder peripheral base address.
- slot – **[out]** Pointer to the variable that stores the enabled slot.

Return values

- kStatus_Success – Successfully get the current active slot.
- kStatus_Fail – Current JPEG decoder is idle, no slot is active.

static inline void JPEGDEC_SetEndian(*JPEG_DECODER_Type* *base, *jpegdec_endian_mode_t* mode)

Sets the JPEG decoder endian mode for decoded data.

Parameters

- base – JPEG decoder peripheral base address.
- mode – Endian mode.

void JPEGDEC_SetJpegBuffer(*jpegdec_decoder_config_t* *config, uint8_t *buffer, size_t length)

Sets the address and length of the JPEG image.

Parameters

- config – **[out]** Decoder configuration.
- data – Start address of the buffer of the raw JPEG image, shall be 8-byte aligned.
- length – Size of the buffer.

void JPEGDEC_SetOutputBuffer(*jpegdec_decoder_config_t* *config, uint8_t *imageData0, uint8_t *imageData1)

Sets the address of the decoded data.

Parameters

- config – **[out]** Decoder configuration.
- imageData0 – Address of the output buffer0, shall be 16-byte aligned.
- imageData1 – Address of the output buffer1, only used when output format is YUV420, shall be 16-byte aligned.

status_t JPEGDEC_ParseHeader(*jpegdec_decoder_config_t* *config)

Parses the JPEG header and stores the info in the decoder configuration structure.

Parameters

- config – **[out]** Decoder configuration.

Return values

- kStatus_Success – Header parsing success.
- kStatus_Fail – JPEG header parsing failed due corrupted header.
- kStatus_JPEGDEC_NotSupported – Header parsing failed due to the image pixel format is not supported or image width/height is larger than 8k or image width/height is not 8-byte aligned.

void JPEGDEC_SetDecodeOption(*jpegdec_decoder_config_t* *config, uint16_t pitch, bool clearStreamBuf, bool autoStart)

Sets the decode option.

Parameters

- config – **[out]** Decoder configuration.
- pitch – Output buffer pitch.
- clearStreamBuf – Set to true to clear the saved bit stream buffer, not used in single frame mode.
- autoStart – Set to true to automatically start the decoding when this descriptor is switched on, not used in single frame mode.

void JPEGDEC_ConfigDecoder(*JPEG_DECODER_Type* *base, const *jpegdec_decoder_config_t* *config)

Sets JPEG decoder decoder configuration.

Call this API to set the decoder configuration and then call *JPEGDEC_StartDecode* to start decode. Need to enable at least one slot in *jpegdec_config_t* first.

Parameters

- base – JPEG decoder peripheral base address.
- config – Decoder configuration.

void JPEGDEC_SetSlotNextDescpt(*JPEG_DECODER_Type* *base, uint8_t slot, *jpegdec_descpt_t* *descriptor)

Sets the address of the next decoder descriptor for certain slot.

Call this API to set the decoder descriptor for certain slot, and then call *JPEGDEC_EnableSlotNextDescpt* to enable the descriptor. If the decode configuration of this descriptor has enabled autoStart then decode will start automatically. Otherwise user has to call *JPEGDEC_StartDecode* manually. Need to enable the slot in *jpegdec_config_t* first.

Parameters

- base – JPEG decoder peripheral base address.
- slot – Slot number.
- descriptor – Pointer to the descriptor structure.

static inline void JPEGDEC_EnableSlotNextDescpt(*JPEG_DECODER_Type* *base, uint8_t slot)

Enables the next decoder descriptor for certain slot.

Call this API after calling JPEGDEC_SetSlotNextDescpt to set valid descriptor for certain slot. User can also call *JPEGDEC_EnableLinkedDescpt* before calling *JPEGDEC_SetSlotNextDescpt*, then the descriptor will be loaded automatically.

Parameters

- base – JPEG decoder peripheral base address.
- slot – Slot number.

static inline void JPEGDEC_EnableLinkedDescpt(*jpegdec_descpt_t* *descriptor)

Enables the next linked descriptor.

Call this API before calling *JPEGDEC_SetSlotNextDescpt*, then the descriptor will be loaded automatically. Otherwise call *JPEGDEC_EnableSlotNextDescpt* after calling *JPEGDEC_SetSlotNextDescpt*.

Parameters

- descriptor – **[out]** Pointer to the descriptor structure.

void JPEGDEC_DescptReset(*jpegdec_descpt_t* *descriptor)

Sets all fields to default values for the descriptor structure.

Note: This function enables the auto start feature.

Parameters

- descriptor – Pointer to the descriptor structure.

JPEGDEC_Type *core

Pointer to decoder core.

JPGDECWRP_Type *wrapper

Pointer to decoder wrapper.

uint32_t outBufAddr0

Base address for output frame buffer0. 16-byte aligned(4 LSBs are zero).

uint32_t outBufAddr1

Base address for output frame buffer1, only used for YUV420 second plane. 16-byte aligned(4 LSBs are zero).

uint32_t outBufPitch

Output buffer pitch.

uint32_t ___pad0___

Reserved.

uint32_t jpegBufAddr

Base address of the JPEG stream buffer for decoding. 16-byte aligned(4 LSBs are zero).

uint32_t jpegBufSize

Size of the JPEG stream buffer. 1k-byte aligned(10 LSBs are zero).

uint32_t height

Image height in pixel. Max supported is 8K(2000h), must be integer times of 8.

uint32_t ___pad1___

Reserved.

uint32_t width

Image width in pixel. Max supported is 8K(2000h), must be integer times of 8.

uint32_t ___pad2___

Reserved.

uint32_t ___pad3___

Reserved.

uint32_t pixelDepth

Pixel depth, *jpegdec_pixel_depth_t*.

uint32_t pixelFormat

The image pixel format, *jpegdec_pixel_format_t*.

uint32_t clearStreamBuf

Set to true to clear the saved bit stream buffer.

uint32_t autoStart

Set to true to automatically start the decoding when this descriptor is switched on.

uint32_t ___pad4___

Reserved.

uint32_t nextDescptAddr

Address of the next decoding descriptor.

jpegdec_decoder_config_t config

The decoding configuration for this descriptor.

bool enable

Enables the decode IP.

bool enableLp

Enables the core to enter low power mode.

jpegdec_endian_mode_t endianMode
: Endian mode for the souce/decoded image.

uint8_t slots
: Mask of enabled slots *_jpegdec_slots*.

JPEG_ALIGN_SIZE(size, align)

struct JPEG_DECODER_Type
: *#include <fsl_jpegdec.h>* JPEG decoder structure definition.

struct _jpegdec_decoder_config
: *#include <fsl_jpegdec.h>* JPEG decoder configuration.

struct _jpegdec_descpt
: *#include <fsl_jpegdec.h>* JPEG decoder descriptor.

  This structure is used to configure the linked decoding descriptor for one or more slots in Single Bit Stream Repeat mode or Context Switch mode.

struct _jpegdec_config
: *#include <fsl_jpegdec.h>* Configuration for JPEG decoder.

2.44 Common Driver

FSL_COMMON_DRIVER_VERSION
: common driver version.

DEBUG_CONSOLE_DEVICE_TYPE_NONE
: No debug console.

DEBUG_CONSOLE_DEVICE_TYPE_UART
: Debug console based on UART.

DEBUG_CONSOLE_DEVICE_TYPE_LPUART
: Debug console based on LPUART.

DEBUG_CONSOLE_DEVICE_TYPE_LPSCI
: Debug console based on LPSCI.

DEBUG_CONSOLE_DEVICE_TYPE_USBCDC
: Debug console based on USBCDC.

DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM
: Debug console based on FLEXCOMM.

DEBUG_CONSOLE_DEVICE_TYPE_IUART
: Debug console based on i.MX UART.

DEBUG_CONSOLE_DEVICE_TYPE_VUSART
: Debug console based on LPC_VUSART.

DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART
: Debug console based on LPC_USART.

DEBUG_CONSOLE_DEVICE_TYPE_SWO
: Debug console based on SWO.

DEBUG_CONSOLE_DEVICE_TYPE_QSCI
: Debug console based on QSCI.

MIN(a, b)
: Computes the minimum of *a* and *b*.

MAX(a, b)
: Computes the maximum of *a* and *b*.

UINT16_MAX
: Max value of uint16_t type.

UINT32_MAX
: Max value of uint32_t type.

UINTPTR_SIZE

UINT64_H(X)
: Macro to get upper 32 bits of a 64-bit value

UINT64_L(X)
: Macro to get lower 32 bits of a 64-bit value

MCUX_MASK_INVERT_8(mask)
: 8-bit mask inversion.

MCUX_MASK_INVERT_16(mask)
: 16-bit mask inversion.

MCUX_MASK_INVERT_32(mask)
: 32-bit mask inversion for completeness.

MCUX_REG_WRITE8(reg, value)
: 8-bit register write macro

MCUX_REG_WRITE16(reg, value)
: 16-bit register write macro

MCUX_REG_WRITE32(reg, value)
: 32-bit register write macro

MCUX_REG_READ8(reg)
: 8-bit register read macro

MCUX_REG_READ16(reg)
: 16-bit register read macro

MCUX_REG_READ32(reg)
: 32-bit register read macro

MCUX_REG_BIT_SET8(reg, mask)
: 8-bit register bit set macro

MCUX_REG_BIT_SET16(reg, mask)
: 16-bit register bit set macro

MCUX_REG_BIT_SET32(reg, mask)
: 32-bit register bit set macro

MCUX_REG_BIT_CLEAR8(reg, mask)
: 8-bit register bit clear macro

MCUX_REG_BIT_CLEAR16(reg, mask)
: 16-bit register bit clear macro

MCUX_REG_BIT_CLEAR32(reg, mask)
: 32-bit register bit clear macro

MCUX_REG_BIT_GET8(reg, mask)
: 8-bit register bit get macro

MCUX_REG_BIT_GET16(reg, mask)
: 16-bit register bit get macro

MCUX_REG_BIT_GET32(reg, mask)
: 32-bit register bit get macro

MCUX_REG_MODIFY8(reg, mask, value)
: 32-bit register read-modify-write macro

MCUX_REG_MODIFY16(reg, mask, value)
: 16-bit register read-modify-write macro

MCUX_REG_MODIFY32(reg, mask, value)
: 32-bit register read-modify-write macro

USEC_TO_COUNT(us, clockFreqInHz)
: Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)
: Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)
: Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)
: Macro to convert a raw count value to millisecond

USEC_TO_COUNT(us, clockFreqInHz)
: Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)
: Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)
: Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)
: Macro to convert a raw count value to millisecond

USEC_TO_COUNT(us, clockFreqInHz)
: Macro to convert a microsecond period to raw count value

COUNT_TO_USEC(count, clockFreqInHz)
: Macro to convert a raw count value to microsecond

MSEC_TO_COUNT(ms, clockFreqInHz)
: Macro to convert a millisecond period to raw count value

COUNT_TO_MSEC(count, clockFreqInHz)
: Macro to convert a raw count value to millisecond

SDK_ISR_EXIT_BARRIER

SDK_ISR_EXIT_BARRIER

SDK_ISR_EXIT_BARRIER

SDK_ALIGN(var, alignbytes)
: Macro to define a variable with alignbytes alignment

SDK_SIZEALIGN(var, alignbytes)
: Macro to change a value to a given size aligned value (rounded up)

  Macro to define a variable with L2 cache line size alignment

  Macro to change a value to a given size aligned value (rounded up)

SDK_SIZEALIGN_UP(var, alignbytes)
: Macro to change a value to a given size aligned value (rounded up), the wrapper of SDK_SIZEALIGN

SDK_SIZEALIGN_DOWN(var, alignbytes)
: Macro to change a value to a given size aligned value (rounded down)

SDK_IS_ALIGNED(var, alignbytes)
: Macro to check if a value is aligned to a given size

SDK_ALIGN(var, alignbytes)
: Macro to define a variable with alignbytes alignment

SDK_L1DCACHE_ALIGN(var)
: Macro to define a variable with L1 d-cache line size alignment

SDK_SIZEALIGN(var, alignbytes)
: Macro to define a variable with L2 cache line size alignment

  Macro to change a value to a given size aligned value (rounded up)

SDK_SIZEALIGN_UP(var, alignbytes)
: Macro to change a value to a given size aligned value (rounded up), the wrapper of SDK_SIZEALIGN

SDK_SIZEALIGN_DOWN(var, alignbytes)
: Macro to change a value to a given size aligned value (rounded down)

SDK_IS_ALIGNED(var, alignbytes)
: Macro to check if a value is aligned to a given size

SDK_ALIGN(var, alignbytes)
: Macro to define a variable with alignbytes alignment

SDK_L1DCACHE_ALIGN(var)
: Macro to define a variable with L1 d-cache line size alignment

SDK_SIZEALIGN(var, alignbytes)
: Macro to define a variable with L2 cache line size alignment

  Macro to change a value to a given size aligned value (rounded up)

SDK_SIZEALIGN_UP(var, alignbytes)
: Macro to change a value to a given size aligned value (rounded up), the wrapper of SDK_SIZEALIGN

SDK_SIZEALIGN_DOWN(var, alignbytes)
: Macro to change a value to a given size aligned value (rounded down)

SDK_IS_ALIGNED(var, alignbytes)
: Macro to check if a value is aligned to a given size

MCUX_DEPRECATED
: Deprecated APIs.

MCUX_DEPRECATED_MACRO
: Deprecated macros.

MCUX_EXPERIMENTAL
: Experimental APIs.

MCUX_EXPERIMENTAL_MACRO
: Experimental macros.

MCUX_DEPRECATED
: Deprecated APIs.

MCUX_DEPRECATED_MACRO
: Deprecated macros.

MCUX_EXPERIMENTAL
: Experimental APIs.

MCUX_EXPERIMENTAL_MACRO
: Experimental macros.

MCUX_DEPRECATED
: Deprecated APIs.

MCUX_DEPRECATED_MACRO
: Deprecated macros.

MCUX_EXPERIMENTAL
: Experimental APIs.

MCUX_EXPERIMENTAL_MACRO
: Experimental macros.

AT_NONCACHEABLE_SECTION_INIT(var)

Define a variable *var* with initial value, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION(var)

Define a variable *var*, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN_INIT(var, alignbytes)

Define a variable *var* with initial value, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_NONCACHEABLE_SECTION_ALIGN(var, alignbytes)

Define a variable *var*, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_NONCACHEABLE_SECTION(var)

Define a variable *var*, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN(var, alignbytes)

Define a variable *var*, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

AT_NONCACHEABLE_SECTION_INIT(var)

Define a variable *var* with initial value, and place it in non-cacheable section.

AT_NONCACHEABLE_SECTION_ALIGN_INIT(var, alignbytes)

Define a variable *var* with initial value, and place it in non-cacheable section, the start address of the variable is aligned to *alignbytes*.

MCUX_CS

AT_QUICKACCESS_SECTION_CODE(func)

Place function in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA(func)

Place data in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_CODE(func)

Place function in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA(var)

Place data in a section which can be accessed quickly by core.

AT_QUICKACCESS_SECTION_DATA_ALIGN(var, alignbytes)

Place data in a section which can be accessed quickly by core, and the variable address is set to align with *alignbytes*.

SDK_ATOMIC_LOCAL_ADD(addr, val)

Add value *val* from the variable at address *address*.

SDK_ATOMIC_LOCAL_SUB(addr, val)

Subtract value *val* to the variable at address *address*.

SDK_ATOMIC_LOCAL_SET(addr, bits)

Set the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR(addr, bits)

Clear the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)

Toggle the bits specifiled by *bits* to the variable at address *address*.

SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)

For the variable at address *address*, clear the bits specifiled by *clearBits* and set the bits specifiled by *setBits*.

SDK_ATOMIC_LOCAL_COMPARE_AND_SET(addr, expected, newValue)

For the variable at address *address*, check whether the value equal to *expected*. If value same as *expected* then update *newValue* to address and return **true** , else return **false** .

SDK_ATOMIC_LOCAL_TEST_AND_SET(addr, newValue)

For the variable at address *address*, set as *newValue* value and return old value.

AT_CACHE_LINE_SECTION(var)

Define a variable *var*, which is cache line size aligned and be placed in CacheLineData section.

AT_CACHE_LINE_SECTION_INIT(var)

Define a variable *var* with initial value, which is cache line size aligned and be placed in CacheLineData.init section.

CACHE_LINE_DATA

MCUX_RAMFUNC

Function attribute to place function in RAM. For example, to place function my_func in ram, use like:

```
MCUX_RAMFUNC my_func
```

RAMFUNCTION_SECTION_CODE(func)

Place function in ram.

enum _status_groups

Status group numbers.

Values:

enumerator kStatusGroup_Generic

Group number for generic status codes.

enumerator kStatusGroup_FLASH

Group number for FLASH status codes.

enumerator kStatusGroup_LPSPI

Group number for LPSPI status codes.

enumerator kStatusGroup_FLEXIO_SPI

Group number for FLEXIO SPI status codes.

enumerator kStatusGroup_DSPI

Group number for DSPI status codes.

enumerator kStatusGroup_FLEXIO_UART

Group number for FLEXIO UART status codes.

enumerator kStatusGroup_FLEXIO_I2C
Group number for FLEXIO I2C status codes.

enumerator kStatusGroup_LPI2C
Group number for LPI2C status codes.

enumerator kStatusGroup_UART
Group number for UART status codes.

enumerator kStatusGroup_I2C
Group number for UART status codes.

enumerator kStatusGroup_LPSCI
Group number for LPSCI status codes.

enumerator kStatusGroup_LPUART
Group number for LPUART status codes.

enumerator kStatusGroup_SPI
Group number for SPI status code.

enumerator kStatusGroup_XRDC
Group number for XRDC status code.

enumerator kStatusGroup_SEMA42
Group number for SEMA42 status code.

enumerator kStatusGroup_SDHC
Group number for SDHC status code

enumerator kStatusGroup_SDMMC
Group number for SDMMC status code

enumerator kStatusGroup_SAI
Group number for SAI status code

enumerator kStatusGroup_MCG
Group number for MCG status codes.

enumerator kStatusGroup_SCG
Group number for SCG status codes.

enumerator kStatusGroup_SDSPI
Group number for SDSPI status codes.

enumerator kStatusGroup_FLEXIO_I2S
Group number for FLEXIO I2S status codes

enumerator kStatusGroup_FLEXIO_MCULCD
Group number for FLEXIO LCD status codes

enumerator kStatusGroup_FLASHIAP
Group number for FLASHIAP status codes

enumerator kStatusGroup_FLEXCOMM_I2C
Group number for FLEXCOMM I2C status codes

enumerator kStatusGroup_I2S
Group number for I2S status codes

enumerator kStatusGroup_IUART
Group number for IUART status codes

enumerator kStatusGroup_CSI
Group number for CSI status codes

enumerator kStatusGroup_MIPI_DSI
Group number for MIPI DSI status codes

enumerator kStatusGroup_SDRAMC
Group number for SDRAMC status codes.

enumerator kStatusGroup_POWER
Group number for POWER status codes.

enumerator kStatusGroup_ENET
Group number for ENET status codes.

enumerator kStatusGroup_PHY
Group number for PHY status codes.

enumerator kStatusGroup_TRGMUX
Group number for TRGMUX status codes.

enumerator kStatusGroup_SMARTCARD
Group number for SMARTCARD status codes.

enumerator kStatusGroup_LMEM
Group number for LMEM status codes.

enumerator kStatusGroup_QSPI
Group number for QSPI status codes.

enumerator kStatusGroup_DMA
Group number for DMA status codes.

enumerator kStatusGroup_EDMA
Group number for EDMA status codes.

enumerator kStatusGroup_DMAMGR
Group number for DMAMGR status codes.

enumerator kStatusGroup_FLEXCAN
Group number for FlexCAN status codes.

enumerator kStatusGroup_LTC
Group number for LTC status codes.

enumerator kStatusGroup_FLEXIO_CAMERA
Group number for FLEXIO CAMERA status codes.

enumerator kStatusGroup_LPC_SPI
Group number for LPC_SPI status codes.

enumerator kStatusGroup_LPC_USART
Group number for LPC_USART status codes.

enumerator kStatusGroup_DMIC
Group number for DMIC status codes.

enumerator kStatusGroup_SDIF
Group number for SDIF status codes.

enumerator kStatusGroup_SPIFI
Group number for SPIFI status codes.

enumerator kStatusGroup_OTP
Group number for OTP status codes.

enumerator kStatusGroup_MCAN
Group number for MCAN status codes.

enumerator kStatusGroup_CAAM
Group number for CAAM status codes.

enumerator kStatusGroup_ECSPI
Group number for ECSPI status codes.

enumerator kStatusGroup_USDHC
Group number for USDHC status codes.

enumerator kStatusGroup_LPC_I2C
Group number for LPC_I2C status codes.

enumerator kStatusGroup_DCP
Group number for DCP status codes.

enumerator kStatusGroup_MSCAN
Group number for MSCAN status codes.

enumerator kStatusGroup_ESAI
Group number for ESAI status codes.

enumerator kStatusGroup_FLEXSPI
Group number for FLEXSPI status codes.

enumerator kStatusGroup_MMDC
Group number for MMDC status codes.

enumerator kStatusGroup_PDM
Group number for MIC status codes.

enumerator kStatusGroup_SDMA
Group number for SDMA status codes.

enumerator kStatusGroup_ICS
Group number for ICS status codes.

enumerator kStatusGroup_SPDIF
Group number for SPDIF status codes.

enumerator kStatusGroup_LPC_MINISPI
Group number for LPC_MINISPI status codes.

enumerator kStatusGroup_HASHCRYPT
Group number for Hashcrypt status codes

enumerator kStatusGroup_LPC_SPI_SSP
Group number for LPC_SPI_SSP status codes.

enumerator kStatusGroup_I3C
Group number for I3C status codes

enumerator kStatusGroup_LPC_I2C_1
Group number for LPC_I2C_1 status codes.

enumerator kStatusGroup_NOTIFIER
Group number for NOTIFIER status codes.

enumerator kStatusGroup_DebugConsole
Group number for debug console status codes.

enumerator kStatusGroup_SEMC
Group number for SEMC status codes.

enumerator kStatusGroup_ApplicationRangeStart
Starting number for application groups.

enumerator kStatusGroup_IAP
Group number for IAP status codes

enumerator kStatusGroup_SFA
Group number for SFA status codes

enumerator kStatusGroup_SPC
Group number for SPC status codes.

enumerator kStatusGroup_PUF
Group number for PUF status codes.

enumerator kStatusGroup_TOUCH_PANEL
Group number for touch panel status codes

enumerator kStatusGroup_VBAT
Group number for VBAT status codes

enumerator kStatusGroup_XSPI
Group number for XSPI status codes

enumerator kStatusGroup_PNGDEC
Group number for PNGDEC status codes

enumerator kStatusGroup_JPEGDEC
Group number for JPEGDEC status codes

enumerator kStatusGroup_AUDMIX
Group number for AUDMIX status codes

enumerator kStatusGroup_HAL_GPIO
Group number for HAL GPIO status codes.

enumerator kStatusGroup_HAL_UART
Group number for HAL UART status codes.

enumerator kStatusGroup_HAL_TIMER
Group number for HAL TIMER status codes.

enumerator kStatusGroup_HAL_SPI
Group number for HAL SPI status codes.

enumerator kStatusGroup_HAL_I2C
Group number for HAL I2C status codes.

enumerator kStatusGroup_HAL_FLASH
Group number for HAL FLASH status codes.

enumerator kStatusGroup_HAL_PWM
Group number for HAL PWM status codes.

enumerator kStatusGroup_HAL_RNG
Group number for HAL RNG status codes.

enumerator kStatusGroup_HAL_I2S
Group number for HAL I2S status codes.

enumerator kStatusGroup_HAL_ADC_SENSOR
Group number for HAL ADC SENSOR status codes.

enumerator kStatusGroup_TIMERMANAGER
Group number for TiMER MANAGER status codes.

enumerator kStatusGroup_SERIALMANAGER
Group number for SERIAL MANAGER status codes.

enumerator kStatusGroup_LED
Group number for LED status codes.

enumerator kStatusGroup_BUTTON
Group number for BUTTON status codes.

enumerator kStatusGroup_EXTERN_EEPROM
Group number for EXTERN EEPROM status codes.

enumerator kStatusGroup_SHELL
Group number for SHELL status codes.

enumerator kStatusGroup_MEM_MANAGER
Group number for MEM MANAGER status codes.

enumerator kStatusGroup_LIST
Group number for List status codes.

enumerator kStatusGroup_OSA
Group number for OSA status codes.

enumerator kStatusGroup_COMMON_TASK
Group number for Common task status codes.

enumerator kStatusGroup_MSG
Group number for messaging status codes.

enumerator kStatusGroup_SDK_OCOTP
Group number for OCOTP status codes.

enumerator kStatusGroup_SDK_FLEXSPINOR
Group number for FLEXSPINOR status codes.

enumerator kStatusGroup_CODEC
Group number for codec status codes.

enumerator kStatusGroup_ASRC
Group number for codec status ASRC.

enumerator kStatusGroup_OTFAD
Group number for codec status codes.

enumerator kStatusGroup_SDIOSLV
Group number for SDIOSLV status codes.

enumerator kStatusGroup_MECC
Group number for MECC status codes.

enumerator kStatusGroup_ENET_QOS
Group number for ENET_QOS status codes.

enumerator kStatusGroup_LOG
Group number for LOG status codes.

enumerator kStatusGroup_I3CBUS
Group number for I3CBUS status codes.

enumerator kStatusGroup_QSCI
Group number for QSCI status codes.

enumerator kStatusGroup_ELEMU
Group number for ELEMU status codes.

enumerator kStatusGroup_QUEUEDSPI
Group number for QSPI status codes.

enumerator kStatusGroup_POWER_MANAGER
Group number for POWER_MANAGER status codes.

enumerator kStatusGroup_IPED
Group number for IPED status codes.

enumerator kStatusGroup_ELS_PKC
Group number for ELS PKC status codes.

enumerator kStatusGroup_CSS_PKC
Group number for CSS PKC status codes.

enumerator kStatusGroup_HOSTIF
Group number for HOSTIF status codes.

enumerator kStatusGroup_CLIF
Group number for CLIF status codes.

enumerator kStatusGroup_BMA
Group number for BMA status codes.

enumerator kStatusGroup_NETC
Group number for NETC status codes.

enumerator kStatusGroup_ELE
Group number for ELE status codes.

enumerator kStatusGroup_GLIKEY
Group number for GLIKEY status codes.

enumerator kStatusGroup_AON_POWER
Group number for AON_POWER status codes.

enumerator kStatusGroup_AON_COMMON
Group number for AON_COMMON status codes.

enumerator kStatusGroup_ENDAT3
Group number for ENDAT3 status codes.

enumerator kStatusGroup_HIPERFACE
Group number for HIPERFACE status codes.

enumerator kStatusGroup_NPX
Group number for NPX status codes.

enumerator kStatusGroup_ELA_CSEC
Group number for ELA_CSEC status codes.

enumerator kStatusGroup_FLEXIO_T_FORMAT
Group number for T-format status codes.

enumerator kStatusGroup_FLEXIO_A_FORMAT
Group number for A-format status codes.

enumerator kStatusGroup_LPC_QSPI
Group number for LPC QSPI status codes.

enumerator kStatusGroup_EVENT_CTRL
Group number for Event controller status codes.

Generic status return codes.

Values:

enumerator kStatus_Success
Generic status for Success.

enumerator kStatus_Fail
Generic status for Fail.

enumerator kStatus_ReadOnly
Generic status for read only failure.

enumerator kStatus_OutOfRange
Generic status for out of range access.

enumerator kStatus_InvalidArgument
Generic status for invalid argument check.

enumerator kStatus_Timeout
Generic status for timeout.

enumerator kStatus_NoTransferInProgress
Generic status for no transfer in progress.

enumerator kStatus_Busy
Generic status for module is busy.

enumerator kStatus_NoData
Generic status for no data is found for the operation.

typedef int32_t status_t
Type used for all status and error return values.

void *SDK_Malloc(size_t size, size_t alignbytes)
Allocate memory with given alignment and aligned size.

This is provided to support the dynamically allocated memory used in cache-able region.

Parameters

- size – The length required to malloc.
- alignbytes – The alignment size.

Return values
The – allocated memory.

void SDK_Free(void *ptr)

Free memory.

Parameters

- ptr – The memory to be release.

void SDK_DelayAtLeastUs(uint32_t delayTime_us, uint32_t coreClock_Hz)

Delay at least for some time. Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

- delayTime_us – Delay time in unit of microsecond.
- coreClock_Hz – Core clock frequency with Hz.

static inline *status_t* EnableIRQ(IRQn_Type interrupt)

Enable specific interrupt.

Empty function for build compatibility.

Parameters

- interrupt – The IRQ number.

Returns

Always return kStatus_Success.

static inline *status_t* DisableIRQ(IRQn_Type interrupt)

Disable specific interrupt.

Empty function for build compatibility.

Parameters

- interrupt – The IRQ number.

Returns

Always return kStatus_Success.

static inline uint32_t DisableGlobalIRQ(void)

Disable the global IRQ.

Disable the global IRQ.

Returns

The global IRQ control register value before been disabled.

static inline void EnableGlobalIRQ(uint32_t intCtrl)

Enable the global IRQ.

Enable the global interrupt by setting the global IRQ control register. Generally this value is the return value of DisableGlobalIRQ.

Parameters

- intCtrl – The global interrupt control register value.

static inline void EnableMachineModeInt(ezhv_mie_mask_t mask)

Enable M-mode interrupt.

Parameters

- mask – M-mode interrupt mask.

static inline void DisableMachineModeInt(ezhv_mie_mask_t mask)

Disable M-mode interrupt.

Parameters

- mask – M-mode interrupt mask.

static inline *status_t* EnableIRQWithPriority(IRQn_Type interrupt, uint8_t priNum)

Enable the IRQ, and also set the interrupt priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to Enable.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_SetPriority(IRQn_Type interrupt, uint8_t priNum)

Set the IRQ priority.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The IRQ to set.
- priNum – Priority number set to interrupt controller register.

Return values

- kStatus_Success – Interrupt priority set successfully
- kStatus_Fail – Failed to set the interrupt priority.

static inline *status_t* IRQ_ClearPendingIRQ(IRQn_Type interrupt)

Clear the pending IRQ flag.

Only handle LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

- interrupt – The flag which IRQ to clear.

Return values

- kStatus_Success – Interrupt priority set successfully

- kStatus_Fail – Failed to set the interrupt priority.

static inline bool _SDK_AtomicLocalCompareAndSet(uint32_t *addr, uint32_t expected, uint32_t newValue)

static inline uint32_t _SDK_AtomicTestAndSet(uint32_t *addr, uint32_t newValue)

FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ

Macro to use the default weak IRQ handler in drivers.

MAKE_STATUS(group, code)

Construct a status code value from a group and code number.

MAKE_VERSION(major, minor, bugfix)

Construct the version number for drivers.

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

```
| Unused    || Major Version || Minor Version ||  Bug Fix    |
31       25  24          17  16          9  8           0
```

ARRAY_SIZE(x)

Computes the number of elements in an array.

SUPPRESS_FALL_THROUGH_WARNING()

For switch case code block, if case section ends without "break;" statement, there wil be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc. To suppress this warning, "SUPPRESS_FALL_THROUGH_WARNING();" need to be added at the end of each case section which misses "break;"statement.

MSDK_REG_SECURE_ADDR(x)

Convert the register address to the one used in secure mode.

MSDK_REG_NONSECURE_ADDR(x)

Convert the register address to the one used in unsecure mode.

Convert the register address to the one used in non-secure mode.

NVIC_SetPriorityGrouping(value)

NVIC_GetPriorityGrouping()

NVIC_SetPriority(irq, value)

NVIC_GetPriority(irq)

NVIC_EnableIRQ(value)

NVIC_GetEnableIRQ(value)

NVIC_DisableIRQ(value)

NVIC_GetPendingIRQ(value)

NVIC_SetPendingIRQ(value)

NVIC_ClearPendingIRQ(value)

NVIC_GetActive(value)

MSDK_REG_SECURE_ADDR(x)

Convert the register address to the one used in secure mode.

MSDK_REG_NONSECURE_ADDR(x)
Convert the register address to the one used in non-secure mode.

MSDK_REG_SECURE_ADDR(x)
Convert the register address to the one used in secure mode.

MSDK_REG_NONSECURE_ADDR(x)
Convert the register address to the one used in non-secure mode.

MSDK_HAS_DWT_CYCCNT
The chip supports DWT CYCCNT or not.

MSDK_INVALID_IRQ_HANDLER
Invalid IRQ handler address.

2.45 LCDIF: LCD interface

status_t LCDIF_Init(LCDIF_Type *base)
Initialize the LCDIF.

This function initializes the LCDIF to work.

Parameters

- base – LCDIF peripheral base address.

Return values

kStatus_Success – Initialize successfully.

void LCDIF_Deinit(LCDIF_Type *base)
De-initialize the LCDIF.

This function disables the LCDIF peripheral clock.

Parameters

- base – LCDIF peripheral base address.

void LCDIF_DpiModeGetDefaultConfig(*lcdif_dpi_config_t* *config)
Get the default configuration for to initialize the LCDIF.

The default configuration value is:

```
config->panelWidth = 0;
config->panelHeight = 0;
config->hsw = 0;
config->hfp = 0;
config->hbp = 0;
config->vsw = 0;
config->vfp = 0;
config->vbp = 0;
config->polarityFlags = kLCDIF_VsyncActiveLow | kLCDIF_HsyncActiveLow | kLCDIF_
↪DataEnableActiveHigh |
kLCDIF_DriveDataOnFallingClkEdge; config->format = kLCDIF_Output24Bit;
```

Parameters

- config – Pointer to the LCDIF configuration.

status_t LCDIF_DpiModeSetConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_dpi_config_t* *config)

Initialize the LCDIF to work in DPI mode.

This function configures the LCDIF DPI display.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

Return values

- kStatus_Success – Initialize successfully.
- kStatus_InvalidArgument – Initialize failed because of invalid argument.

status_t LCDIF_DbiModeSetConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_dbi_config_t* *config)

Initialize the LCDIF to work in DBI mode.

This function configures the LCDIF DBI display.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

Return values

- kStatus_Success – Initialize successfully.
- kStatus_InvalidArgument – Initialize failed because of invalid argument.

void LCDIF_DbiModeGetDefaultConfig(*lcdif_dbi_config_t* *config)

Get the default configuration to initialize the LCDIF DBI mode.

The default configuration value is:

```
config->swizzle         = kLCDIF_DbiOutSwizzleRGB;
config->format          = kLCDIF_DbiOutD8RGB332;
config->acTimeUnit      = 0;
config->type            = kLCDIF_DbiTypeA_ClockedE;
config->reversePolarity = false;
config->writeWRPeriod   = 3U;
config->writeWRAssert   = 0U;
config->writeCSAssert   = 0U;
config->writeWRDeassert = 0U;
config->writeCSDeassert = 0U;
config->typeCTas        = 1U;
config->typeCSCLTwrl    = 1U;
config->typeCSCLTwrh    = 1U;
```

Parameters

- config – Pointer to the LCDIF DBI configuration.

static inline void LCDIF_DbiReset(LCDIF_Type *base, uint8_t displayIndex)

Reset the DBI module.

Parameters

- displayIndex – Display index.

- base – LCDIF peripheral base address.

static inline bool LCDIF_DbiIsTypeCFifoFull(LCDIF_Type *base, uint8_t displayIndex)

Check whether the FIFO is full in DBI mode type C.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.

Return values

- true – FIFO full.
- false – FIFO not full.

void LCDIF_DbiSelectArea(LCDIF_Type *base, uint8_t displayIndex, uint16_t startX, uint16_t startY, uint16_t endX, uint16_t endY, bool isTiled)

Select the update area in DBI mode.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- startX – X coordinate for start pixel.
- startY – Y coordinate for start pixel.
- endX – X coordinate for end pixel.
- endY – Y coordinate for end pixel.
- isTiled – true if the pixel data is tiled.

static inline void LCDIF_DbiSendCommand(LCDIF_Type *base, uint8_t displayIndex, uint8_t cmd)

Send command to DBI port.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- cmd – the DBI command to send.

void LCDIF_DbiSendData(LCDIF_Type *base, uint8_t displayIndex, const uint8_t *data, uint32_t dataLen_Byte)

brief Send data to DBI port.

Can be used to send light weight data to panel. To send pixel data in frame buffer, use *LCDIF_DbiWriteMem*.

param base LCDIF peripheral base address. param displayIndex Display index. param data pointer to data buffer. param dataLen_Byte data buffer length in byte.

void LCDIF_DbiSendCommandAndData(LCDIF_Type *base, uint8_t displayIndex, uint8_t cmd, const uint8_t *data, uint32_t dataLen_Byte)

Send command followed by data to DBI port.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- cmd – the DBI command to send.
- data – pointer to data buffer.

- dataLen_Byte – data buffer length in byte.

static inline void LCDIF_DbiWriteMem(LCDIF_Type *base, uint8_t displayIndex)

Send pixel data in frame buffer to panel controller memory.

This function starts sending the pixel data in frame buffer to panel controller, user can monitor interrupt *kLCDIF_Display0FrameDoneInterrupt* to know when then data sending finished.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.

void LCDIF_SetFrameBufferConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_fb_config_t* *config)

Configure the LCDIF frame buffer.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to configure the legacy video layer, and use LCDIF_SetOverlayFrameBufferConfig to configure the overlay layers.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

void LCDIF_FrameBufferGetDefaultConfig(*lcdif_fb_config_t* *config)

Get default frame buffer configuration.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to get the default configuration for all the 3 layers.

The default configuration is

```
config->enable = true;
config->enableGamma = false;
config->format = kLCDIF_PixelFormatRGB565;
```

Parameters

- config – Pointer to the configuration structure.

static inline void LCDIF_SetFrameBufferAddr(LCDIF_Type *base, uint8_t displayIndex, uint32_t address)

Set the frame buffer to LCDIF.

Note: The address must be 128 bytes aligned.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- address – Frame buffer address.

void LCDIF_SetFrameBufferStride(LCDIF_Type *base, uint8_t displayIndex, uint32_t strideBytes)

Set the frame buffer stride.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- strideBytes – The stride in byte.

static inline void LCDIF_SetFrameBufferUVAddr(LCDIF_Type *base, uint8_t displayIndex, uint32_t address)

Set the frame buffer to LCDIF for UV plane when the input format is YUV420.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- address – Frame buffer address.

static inline void LCDIF_SetFrameBufferUVStride(LCDIF_Type *base, uint8_t displayIndex, uint32_t strideBytes)

void LCDIF_SetFrameBufferPosition(LCDIF_Type *base, uint8_t displayIndex, uint16_t topLeftX, uint16_t topLeftY, uint16_t width, uint16_t height)

Configure the video layer position.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to configure the legacy video layer, and use *LCDIF_SetOverlayLayerPosition* to configure the overlay layers.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- topLeftX – The x value of thr top-left coordinate.
- topLeftY – The y value of thr top-left coordinate.
- width – The width of the layer.
- height – The height of the layer.

void LCDIF_SetOverlayLayerConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_fb_config_t* *config, uint8_t layerIndex)

Configure the overlay layers for LCDIF frame buffer.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to configure the overlay layers, and use *LCDIF_SetFrameBufferConfig* to configure the legacy video layer.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.
- layerIndex – Pointer to the configuration structure.

void LCDIF_SetOverlayLayerPosition(LCDIF_Type *base, uint8_t displayIndex, uint16_t topLeftX, uint16_t topLeftY, uint16_t width, uint16_t height, uint8_t layerIndex)

Configure the overlay layer position.

@Note: For LCDIF of version DC8000 there can be 3 layers in the pre-processing, compared with the older version. Apart from the video layer, there are also 2 overlay layers which shares the same configurations. Use this API to configure the overlay layers, and use *LCDIF_SetFrameBufferPosition* to configure the legacy video layer.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- topLeftX – The x value of thr top-left coordinate.
- topLeftY – The y value of thr top-left coordinate.
- width – The width of the layer.
- height – The height of the layer.
- layerIndex – Pointer to the configuration structure.

void LCDIF_SetOverlayLayerAddr(LCDIF_Type *base, uint8_t displayIndex, uint32_t address, uint8_t layerIndex)

Sets the frame buffer address for overlay layer.

Note: The address must be 64 bytes aligned.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- address – Frame buffer address.
- layerIndex – Which layer to configure.

void LCDIF_SetOverlayLayerStride(LCDIF_Type *base, uint8_t displayIndex, uint32_t strideBytes, uint8_t layerIndex)

Sets the frame buffer stride for overlay layer.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- strideBytes – The stride in byte.
- layerIndex – Which layer to configure.

static inline void LCDIF_SetOverlayLayerUVAddr(LCDIF_Type *base, uint8_t displayIndex, uint32_t address)

Sets the frame buffer address for overlay layer 0 for UV plane when the input format is YUV420.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.

- address – Frame buffer address.

static inline void LCDIF_SetOverlayLayerUVStride(LCDIF_Type *base, uint8_t displayIndex, uint32_t strideBytes)

static inline void LCDIF_SetFrameBufferBackground(LCDIF_Type *base, uint8_t displayIndex, uint32_t color)

Sets the color for background layer.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- color – Background color in ARGB8888 format.

status_t LCDIF_GetPorterDuffConfig(*lcdif_porter_duff_blend_mode_t* mode, *lcdif_layer_alpha_blend_config_t* *config)

brief Get the alpha blend configuration by porter duff blend mode.

param mode The blend mode. param config Pointer to the configuration. retval kStatus_Success Successfully get the configuration. retval kStatus_InvalidArgument The blend mode not supported.

void LCDIF_PanelGetDefaultConfig(*lcdif_panel_config_t* *config)

Gets default panel configuration.

The default configuration is

```
config->enable = true;
config->enableGamma = false;
config->order       = kLCDIF_VideoOverlay0Overlay1;
config->endian      = kLCDIF_NoSwap;
```

Parameters

- config – Pointer to the configuration structure.

void LCDIF_SetPanelConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_panel_config_t* *config)

Configure the LCDIF panel.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- config – Pointer to the configuration structure.

static inline void LCDIF_SetLayerOrder(LCDIF_Type *base, uint8_t displayIndex, *lcdif_layer_order_t* order)

Sets the layer order for the color key and alpha blend precess.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- order – The order of the 3 configurable layers.

static inline void LCDIF_SetEndianMode(LCDIF_Type *base, uint8_t displayIndex, *lcdif_endian_mode_t* mode)

Sets the endian mode.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- mode – The byte order for the endian mode.

static inline void LCDIF_EnableUpdate(LCDIF_Type *base, bool enable)

Enables the module to update the new set of configuration in the next frame's VBLANK.

Parameters

- base – LCDIF peripheral base address.
- enable – True to enable, false to disable.

static inline void LCDIF_EnableGamma(LCDIF_Type *base, bool enable)

Enables the gamma correction.

Parameters

- base – LCDIF peripheral base address.
- enable – True to enable, false to disable.

static inline void LCDIF_EnablePanel(LCDIF_Type *base, bool enable)

Enables the panel output.

When disables, all pixels will be black which allows a panel to have correct timing without any pixels.

Parameters

- base – LCDIF peripheral base address.
- enable – True to enable, false to disable.

static inline void LCDIF_Start(LCDIF_Type *base)

Starts the interface mode transfer. Self-clear.

All registers will be copied to the working set and resets the pixel counter.

Parameters

- base – LCDIF peripheral base address.

static inline void LCDIF_Reset(LCDIF_Type *base)

Resets the module. Self-clear.

Parameters

- base – LCDIF peripheral base address.

static inline void LCDIF_SetUpdateReady(LCDIF_Type *base)

Set the update of the double buffer register ready.

Call this API after updating any double buffered register to let the change take effect in next v-blank.

Parameters

- base – LCDIF peripheral base address.

void LCDIF_SetDitherConfig(LCDIF_Type *base, uint8_t displayIndex, const *lcdif_dither_config_t* *config)

Set the dither configuration.

Parameters

- base – LCDIF peripheral base address.

- displayIndex – Index to configure.
- config – Pointer to the configuration structure.

void LCDIF_SetGammaData(LCDIF_Type *base, uint8_t displayIndex, uint16_t startIndex, const uint32_t *gamma, uint16_t gammaLen)

Set the gamma translation values to the LCDIF gamma table.

Parameters

- base – LCDIF peripheral base address.
- displayIndex – Display index.
- startIndex – Start index in the gamma table that the value will be set to.
- gamma – The gamma values to set to the gamma table in LCDIF, could be defined using LCDIF_MAKE_GAMMA_VALUE.
- gammaLen – The length of the gamma.

static inline void LCDIF_EnableInterrupts(LCDIF_Type *base, uint32_t mask)

Enables LCDIF interrupt requests.

Parameters

- base – LCDIF peripheral base address.
- mask – The interrupts to enable, pass in as OR'ed value of *_lcdif_interrupt*.

static inline void LCDIF_DisableInterrupts(LCDIF_Type *base, uint32_t mask)

Disable LCDIF interrupt requests.

Parameters

- base – LCDIF peripheral base address.
- mask – The interrupts to disable, pass in as OR'ed value of *_lcdif_interrupt*.

static inline uint32_t LCDIF_GetAndClearInterruptPendingFlags(LCDIF_Type *base)

Get and clear LCDIF interrupt pending status.

Note: The interrupt must be enabled, otherwise the interrupt flags will not assert.

Parameters

- base – LCDIF peripheral base address.

Returns

The interrupt pending status.

void LCDIF_CursorGetDefaultConfig(*lcdif_cursor_config_t* *config)

Get the hardware cursor default configuration.

The default configuration values are:

```
config->enable = true;
config->format = kLCDIF_CursorMasked;
config->hotspotOffsetX = 0;
config->hotspotOffsetY = 0;
```

Parameters

- config – Pointer to the hardware cursor configuration structure.

void LCDIF_SetCursorConfig(LCDIF_Type *base, const *lcdif_cursor_config_t* *config)

Configure the cursor.

Parameters

- base – LCDIF peripheral base address.
- config – Cursor configuration.

static inline void LCDIF_SetCursorHotspotPosition(LCDIF_Type *base, uint16_t x, uint16_t y)

Set the cursor hotspot postion.

Parameters

- base – LCDIF peripheral base address.
- x – X coordinate of the hotspot, range 0 ~ 8191.
- y – Y coordinate of the hotspot, range 0 ~ 8191.

static inline void LCDIF_SetCursorBufferAddress(LCDIF_Type *base, uint32_t address)

Set the cursor memory address.

Parameters

- base – LCDIF peripheral base address.
- address – Memory address.

void LCDIF_SetCursorColor(LCDIF_Type *base, uint32_t background, uint32_t foreground)

Set the cursor color.

Parameters

- base – LCDIF peripheral base address.
- background – Background color, could be defined use *LCDIF_MAKE_CURSOR_COLOR*
- foreground – Foreground color, could be defined use *LCDIF_MAKE_CURSOR_COLOR*

FSL_LCDIF_DRIVER_VERSION

enum _lcdif_polarity_flags

LCDIF signal polarity flags.

Values:

enumerator kLCDIF_VsyncActiveLow

VSYNC active low.

enumerator kLCDIF_VsyncActiveHigh

VSYNC active high.

enumerator kLCDIF_HsyncActiveLow

HSYNC active low.

enumerator kLCDIF_HsyncActiveHigh

HSYNC active high.

enumerator kLCDIF_DataEnableActiveLow
Data enable line active low.

enumerator kLCDIF_DataEnableActiveHigh
Data enable line active high.

enumerator kLCDIF_DriveDataOnFallingClkEdge
Drive data on falling clock edge, capture data on rising clock edge.

enumerator kLCDIF_DriveDataOnRisingClkEdge
Drive data on falling clock edge, capture data on rising clock edge.

enum _lcdif_output_format
LCDIF DPI output format.
Values:

enumerator kLCDIF_Output16BitConfig1
16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_GGGBBBBB.

enumerator kLCDIF_Output16BitConfig2
16-bit configuration 2. RGB565: XXXRRRRR_XXGGGGGG_XXXBBBBB.

enumerator kLCDIF_Output16BitConfig3
16-bit configuration 3. RGB565: XXRRRRRX_XXGGGGGG_XXBBBBBX.

enumerator kLCDIF_Output18BitConfig1
18-bit configuration 1. RGB666: XXXXXXRR_RRRRGGGG_GGBBBBBB.

enumerator kLCDIF_Output18BitConfig2
18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBBBBB.

enumerator kLCDIF_Output24Bit
24-bit.

enum _lcdif_fb_format
LCDIF frame buffer pixel format.
Values:

enumerator kLCDIF_PixelFormatARGB4444
ARGB4444, 16-bit each pixel, 4-bit each element.

enumerator kLCDIF_PixelFormatARGB1555
ARGB1555, 16-bit each pixel, 5-bit each element.

enumerator kLCDIF_PixelFormatRGB565
RGB565, 16-bit each pixel.

enumerator kLCDIF_PixelFormatARGB8888
ARGB8888, 32-bit each pixel, 8-bit each element.

enumerator kLCDIF_PixelFormatRGB888
RGB888, 24-bit each pixel, 8-bit each element.

enumerator kLCDIF_PixelFormatARGB8565
ARGB8565, 24-bit each pixel.

enumerator kLCDIF_PixelFormatARGB8888Tiled
ARGB8888, 32-bit each pixel, 8-bit each element in tiled format, not supported in overlay layer 1.

enumerator kLCDIF_PixelFormatYUV422Tiled
YUV422, component order V-Y-U-Y, in tiled format, not supported in overlay layer 1.

enumerator kLCDIF_PixelFormatYUV420Tiled
YUV420, in tiled format, need 2 plane, not supported in overlay layer 1.

enumerator kLCDIF_PixelFormatRGB888Tiled
RGB888, 24-bit each pixel, 8-bit each element in tiled format, not supported in overlay layer 1.

enumerator kLCDIF_PixelFormatARGB8565Tiled
ARGB8565, 24-bit each pixel in tiled format, not supported in overlay layer 1.

enum _lcdif_layer_input_order
LCDIF layer input pixel order.
Values:

enumerator kLCDIF_PixelInputOrderARGB
Input order ARGB.

enumerator kLCDIF_PixelInputOrderRGBA
Input order RGBA.

enumerator kLCDIF_PixelInputOrderABGR
Input order ABGR.

enumerator kLCDIF_PixelInputOrderBGRA
Input order BGRA.

enum _lcdif_layer_rotate_flip
LCDIF layer rotation or flip configuration.
Values:

enumerator kLCDIF_Rotate0
Clock wise rotate 0 degree.

enumerator kLCDIF_Rotate90
Clock wise rotate 90 degree, not supported yet.

enumerator kLCDIF_Rotate180
Clock wise rotate 180 degree.

enumerator kLCDIF_Rotate270
Clock wise rotate 270 degree, not supported yet.

enumerator kLCDIF_FlipHorizontal
Horizontal flip.

enumerator kLCDIF_FlipVertical
Vertical flip.

enum _lcdif_layer_decompress_mode
LCDIF layer input data decompress mode.
Values:

enumerator kLCDIF_DecompressNone
Decompression disabled. The input is straight RGB/YUV data.

enumerator kLCDIF_DecompressDECNanoNoneSample
DECNano decompression with none sample.

enumerator kLCDIF_DecompressDECNanoHSample
DECNano decompression with horizontal sample.

enumerator kLCDIF_DecompressDECNanoHVSample
DECNano decompression with both horizontal and vertical sample.

enumerator kLCDIF_DecompressETC2
ETC2 decompression.

Alpha blend alpha component mode. .
Values:

enumerator kLCDIF_AlphaStraight
Use straight alpha, s0_alpha' = s0_alpha.

enumerator kLCDIF_AlphaInversed
Use inversed alpha, s0_alpha' = 0xFF - s0_alpha.

LCDIF layer global alpha mode for alpha blend. .
Values:

enumerator kLCDIF_AlphaLocal
Use the alpha(or inversed alpha) component directly.

enumerator kLCDIF_AlphaGlobal
Use the global alpha instead.

enumerator kLCDIF_AlphaScaled
Multiply the alpha(or inversed alpha) component by the global alpha.

Alpha blend factor mode. .

Values:

enumerator kLCDIF_AlphaFactorZero
Use 0 as the blending factor.

enumerator kLCDIF_AlphaFactorOne
Use 1 as the blending factor.

enumerator kLCDIF_AlphaFactorStraight
Use the modified alpha component(As" for source layer or Ad" for destination layer) as the blending factor.

enumerator kLCDIF_AlphaFactorInversed
Use the inversed alpha component(1-As" for source layer or 1-Ad" for destination layer) as the blending factor.

enum _lcdif_porter_duff_blend_mode
LCDIF Porter Duff blend mode. Note: don't change the enum item value.

Values:

enumerator kLCDIF_PorterDuffSrc
Source Only

enumerator kLCDIF_PorterDuffAtop
Source Atop

enumerator kLCDIF_PorterDuffOver
Source Over

enumerator kLCDIF_PorterDuffIn
Source In.

enumerator kLCDIF_PorterDuffOut
Source Out.

enumerator kLCDIF_PorterDuffDst
Destination Only.

enumerator kLCDIF_PorterDuffDstAtop
Destination Atop.

enumerator kLCDIF_PorterDuffDstOver
Destination Over.

enumerator kLCDIF_PorterDuffDstIn
Destination In.

enumerator kLCDIF_PorterDuffDstOut
Destination Out.

enumerator kLCDIF_PorterDuffPlus
Clear.

enumerator kLCDIF_PorterDuffXor
XOR.

enumerator kLCDIF_PorterDuffClear
Clear.

enumerator kLCDIF_PorterDuffMax

enum _lcdif_layer_convert_standard
LCDIF Color space convert standard between RGB and YUV when the input format is tiled YUV.

Values:

enumerator kLCDIF_ConvertBT601
Use standard BT601.

enumerator kLCDIF_ConvertBT709
Use standard BT709.

enum _lcdif_layer_order
LCDIF the layer order from bottom to top.

There are 5 layers in total, apart from the background layer fixed in the bottom, and the cursor layer on top that does not participate in the color key and alpha blend, the order of the other 3 layers video/graphic layer, overlay layer 0 and 1 can be changed. During color key or alpha blend process, between 2 adjacent layers, the lower layer is considered as destination layer and the upper layer is considered as source layer. The precess is performed from bottom to top, after processing 2 layes, the generated new layer will be considered as destination layer then participate in the next process with the next upper layer.

Values:

enumerator kLCDIF_VideoOverlay0Overlay1
In the order of video, overlay0, overlay1.

enumerator kLCDIF_VideoOverlay1Overlay0
In the order of video, overlay1, overlay0.

enumerator kLCDIF_Overlay0VideoOverlay1
In the order of overlay0, video, overlay1.

enumerator kLCDIF_Overlay0Overlay1Video
In the order of overlay0, overlay1, video.

enumerator kLCDIF_Overlay1VideoOverlay0
In the order of overlay1, video, overlay0.

enumerator kLCDIF_Overlay1Overlay0Video
In the order of overlay1, overlay0, video.

enum _lcdif_endian_mode
LCDIF the pixel source endian mode.
Values:

enumerator kLCDIF_NoSwap
No swap.

enumerator kLCDIF_HalfWordSwap
16-bit boundary swap, from 0123 to 1032.

enumerator kLCDIF_WordSwap
32-bit boundary swap, from 0123 to 3210.

enum _lcdif_interrupt
LCDIF interrupt and status.
Values:

enumerator kLCDIF_Display0FrameDoneInterrupt
The last pixel of visible area in frame is shown.

enumerator kLCDIF_Display0DbiConfigErrorInterrupt
The DBI command is illegal.

enumerator kLCDIF_PanelUnderflowInterrupt
The AXI clock and pixel clock mismatch causing data supply insufficient.

enumerator kLCDIF_SoftwareResetDoneInterrupt
A software reset has completed.

enumerator kLCDIF_BusErrorInterrupt
An AXI bus transfer error occurrs.

enum _lcdif_cursor_format

LCDIF cursor format.

Values:

enumerator kLCDIF_CursorMasked

Masked format.

enumerator kLCDIF_CursorARGB8888

ARGB8888.

enum _lcdif_dbi_cmd_flag

LCDIF DBI command flag.

Values:

enumerator kLCDIF_DbiCmdAddress

Send address (or command).

enumerator kLCDIF_DbiCmdWriteMem

Start write memory.

enumerator kLCDIF_DbiCmdData

Send data.

enumerator kLCDIF_DbiCmdReadMem

Start read memory.

enum _lcdif_dbi_out_format

LCDIF DBI output format.

Values:

enumerator kLCDIF_DbiOutD8RGB332

8-bit data bus width, pixel RGB332. For type A or B. 1 pixel sent in 1 cycle.

enumerator kLCDIF_DbiOutD8RGB444

8-bit data bus width, pixel RGB444. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF_DbiOutD8RGB565

8-bit data bus width, pixel RGB565. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD8RGB666

8-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 3 cycles, data bus 2 LSB not used.

enumerator kLCDIF_DbiOutD8RGB888

8-bit data bus width, pixel RGB888. For type A or B. 1 pixel sent in 3 cycles.

enumerator kLCDIF_DbiOutD9RGB666

9-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD16RGB332

16-bit data bus width, pixel RGB332. For type A or B. 2 pixels sent in 1 cycle.

enumerator kLCDIF_DbiOutD16RGB444

16-bit data bus width, pixel RGB444. For type A or B. 1 pixel sent in 1 cycle, data bus 4 MSB not used.

enumerator kLCDIF_DbiOutD16RGB565

16-bit data bus width, pixel RGB565. For type A or B. 1 pixel sent in 1 cycle.

enumerator kLCDIF_DbiOutD16RGB666Option1

16-bit data bus width, pixel RGB666. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF_DbiOutD16RGB666Option2

16-bit data bus width, pixel RGB666. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD16RGB888Option1

16-bit data bus width, pixel RGB888. For type A or B. 2 pixels sent in 3 cycles.

enumerator kLCDIF_DbiOutD16RGB888Option2

16-bit data bus width, pixel RGB888. For type A or B. 1 pixel sent in 2 cycles.

enumerator kLCDIF_DbiOutD1RGB565Option1

1-bit data bus width, pixel RGB565. For type C option 1, use extra bit to distinguish Data and Command (DC).

enumerator kLCDIF_DbiOutD1RGB565Option2

1-bit data bus width, pixel RGB565. For type C option 2, use extra byte to distinguish Data and Command (DC).

enumerator kLCDIF_DbiOutD1RGB565Option3

1-bit data bus width, pixel RGB565. For type C option 3, use extra DC line to distinguish Data and Command (DC).

enumerator kLCDIF_DbiOutD1RG888Option1

1-bit data bus width, pixel RGB888. For type C option 1, use extra bit to distinguish Data and Command (DC).

enumerator kLCDIF_DbiOutD1RG888Option2

1-bit data bus width, pixel RGB888. For type C option 2, use extra byte to distinguish Data and Command (DC).

enumerator kLCDIF_DbiOutD1RG888Option3

1-bit data bus width, pixel RGB888. For type C option 3, use extra DC line to distinguish Data and Command (DC).

enum _lcdif_dbi_type

LCDIF DBI type.

Values:

enumerator kLCDIF_DbiTypeA_FixedE
Selects DBI type A fixed E mode, 68000, Motorola mode.

enumerator kLCDIF_DbiTypeA_ClockedE
Selects DBI Type A Clocked E mode, 68000, Motorola mode.

enumerator kLCDIF_DbiTypeB
Selects DBI type B, 8080, Intel mode.

enumerator kLCDIF_DbiTypeC
Selects DBI type C, SPI mode.

enum __lcdif_dbi_out_swizzle
LCDIF DBI output swizzle.

Values:

enumerator kLCDIF_DbiOutSwizzleRGB
RGB

enumerator kLCDIF_DbiOutSwizzleBGR
BGR

typedef enum *_lcdif_output_format* lcdif_output_format_t
LCDIF DPI output format.

typedef struct *_lcdif_dpi_config* lcdif_dpi_config_t
Configuration for LCDIF module to work in DBI mode.

typedef enum *_lcdif_fb_format* lcdif_fb_format_t
LCDIF frame buffer pixel format.

typedef enum *_lcdif_layer_input_order* lcdif_layer_input_order_t
LCDIF layer input pixel order.

typedef enum *_lcdif_layer_rotate_flip* lcdif_layer_rotate_flip_t
LCDIF layer rotation or flip configuration.

typedef struct *_lcdif_layer_colorkey* lcdif_layer_colorkey_t
LCDIF layer color key configuration.

typedef enum *_lcdif_layer_decompress_mode* lcdif_layer_decompress_mode_t
LCDIF layer input data decompress mode.

typedef enum *_lcdif_porter_duff_blend_mode* lcdif_porter_duff_blend_mode_t
LCDIF Porter Duff blend mode. Note: don't change the enum item value.

typedef enum *_lcdif_layer_convert_standard* lcdif_layer_convert_standard_t
LCDIF Color space convert standard between RGB and YUV when the input format is tiled YUV.

typedef struct *_lcdif_fb_config* lcdif_fb_config_t

LCDIF frame buffer configuration.

For LCDIF of DC8000 version, there are 3 layers in the pre-processing, all shares the same frame buffer configuration.

typedef enum *_lcdif_layer_order* lcdif_layer_order_t

LCDIF the layer order from bottom to top.

There are 5 layers in total, apart from the background layer fixed in the bottom, and the cursor layer on top that does not participate in the color key and alpha blend, the order of the other 3 layers video/graphic layer, overlay layer 0 and 1 can be changed. During color key or alpha blend process, between 2 adjacent layers, the lower layer is considered as destination layer and the upper layer is considered as source layer. The precess is performed from bottom to top, after processing 2 layes, the generated new layer will be considered as destination layer then participate in the next process with the next upper layer.

typedef enum *_lcdif_endian_mode* lcdif_endian_mode_t

LCDIF the pixel source endian mode.

typedef struct *_lcdif_panel_config* lcdif_panel_config_t

LCDIF panel configuration.

typedef enum *_lcdif_cursor_format* lcdif_cursor_format_t

LCDIF cursor format.

typedef struct *_lcdif_cursor_config* lcdif_cursor_config_t

LCDIF cursor configuration.

typedef struct *_lcdif_dither_config* lcdif_dither_config_t

LCDIF dither configuration.

a. Decide which bit of pixel color to enhance. This is configured by the lcdif_dither_config_t::redSize, lcdif_dither_config_t::greenSize, and lcdif_dither_config_t::blueSize. For example, setting redSize=6 means it is the 6th bit starting from the MSB that we want to enhance, in other words, it is the RedColor[2]bit from RedColor[7:0]. greenSize and blueSize function in the same way.

b. Create the look-up table. a. The Look-Up Table includes 16 entries, 4 bits for each. b. The Look-Up Table provides a value U[3:0] through the index X[1:0] and Y[1:0]. c. The color value RedColor[3:0] is used to compare with this U[3:0]. d. If RedColor[3:0] > U[3:0], and RedColor[7:2] is not 6'b111111, then the final color value is: NewRedColor = RedColor[7:2] + 1'b1. e. If RedColor[3:0] <= U[3:0], then NewRedColor = RedColor[7:2].

typedef enum *_lcdif_dbi_out_format* lcdif_dbi_out_format_t

LCDIF DBI output format.

typedef enum *_lcdif_dbi_type* lcdif_dbi_type_t

LCDIF DBI type.

typedef enum *_lcdif_dbi_out_swizzle* lcdif_dbi_out_swizzle_t

LCDIF DBI output swizzle.

typedef struct *_lcdif_dbi_config* lcdif_dbi_config_t

LCDIF DBI configuration.

LCDIF_MAKE_CURSOR_COLOR(r, g, b)

Construct the cursor color, every element should be in the range of 0 ~ 255.

LCDIF_MAKE_GAMMA_VALUE(r, g, b)

Construct the gamma value set to LCDIF gamma table, every element should be in the range of 0~255.

LCDIF_MAKE_BACKGROUND_VALUE(a, r, g, b)

Construct the colour value for the background layer.

LCDIF_ALIGN_ADDR(addr, align)

Calculate the aligned address for LCDIF buffer.

LCDIF_FB_ALIGN

The frame buffer should be 128 byte aligned.

LCDIF_FB_UV_ALIGN

LCDIF_GAMMA_INDEX_MAX

Gamma index max value.

LCDIF_CURSOR_SIZE

The cursor size is 32 x 32.

LCDIF_FRAMEBUFFERCONFIG0_OUTPUT_MASK

LCDIF_ADDR_CPU_2_IP(addr)

struct _lcdif_dpi_config

#include <fsl_lcdif.h> Configuration for LCDIF module to work in DBI mode.

Public Members

uint16_t panelWidth

Display panel width, pixels per line.

uint16_t panelHeight

Display panel height, how many lines per panel.

uint8_t hsw

HSYNC pulse width.

uint8_t hfp

Horizontal front porch.

uint8_t hbp

Horizontal back porch.

uint8_t vsw
: VSYNC pulse width.

uint8_t vfp
: Vrtical front porch.

uint8_t vbp
: Vertical back porch.

uint32_t polarityFlags
: OR'ed value of *_lcdif_polarity_flags*, used to contol the signal polarity.

lcdif_output_format_t format
: DPI output format.

struct __lcdif_layer_colorkey
: *#include <fsl_lcdif.h>* LCDIF layer color key configuration.

Public Members

bool enable
: Enable the color keying.

struct lcdif_layer_alpha_blend_config_t
: *#include <fsl_lcdif.h>* LCDIF frame buffer alpha blend configuration.

The alpha blending formula is C = Fs x Cs + Fd x Cd, and A = Fs x As'' + Fd x Ad''. C/A: generated final color/alpha component, Cs/Cd: original color component for source and destination layer, As''/Ad'': modified alpha component for source and destination layer, take As'' as example, As' = As or (1 - As), chosen by srcAlphaMode As'' = As' or Ags or (As' x Ags), chosen by srcGlobalAlphaMode, Ags configured by srcGlobalAlpha Fs/Fd: blending factor for source/destination layer, take Fs as example, Fs = 0 or 1 or As'' or (1 - As''), chosen by srcFactorMode

Public Members

uint32_t enable
: Enables or disables alpha blend configuration.

uint32_t srcAlphaMode
: Use the straight(As) or inversed(1-As) alpha value for modified alpha for the source layer, see *lcdif_layer_alpha_mode_t*.

uint32_t srcGlobalAlphaMode
: Source layer global alpha mode, see *lcdif_layer_global_alpha_mode_t*.

uint32_t srcFactorMode
: Source layer blend factor mode, see *lcdif_layer_alpha_factor_mode_t*.

uint32_t useSrcAlpha

Set to true to use source alpha for the source blending factor, otherwise use the destination alpha.

uint32_t dstAlphaMode

Use the straight(Ad) or inversed(1-Ad) alpha value for modified alpha for the destination layer, see *lcdif_layer_alpha_mode_t*.

uint32_t dstGlobalAlphaMode

Destination layer global alpha mode, see *lcdif_layer_global_alpha_mode_t*.

uint32_t dstFactorMode

Destination layer blend factor mode, see *lcdif_layer_alpha_factor_mode_t*.

uint32_t useDstAlpha

Set to true to use destination alpha for the destination blending factor, otherwise use the source alpha.

uint32_t srcGlobalAlpha

Source layer global alpha value, 0~255.

uint32_t dstGlobalAlpha

Destination layer global alpha value, 0~255.

struct __lcdif_fb_config

#include <fsl_lcdif.h> LCDIF frame buffer configuration.

For LCDIF of DC8000 version, there are 3 layers in the pre-processing, all shares the same frame buffer configuration.

Public Members

bool enable

Enable the layer output.

bool enableClear

Enable the color clear for this layer, onve enabled, the whole layer will be filled with the color configured in clearValue.

uint32_t clearValue

The value used for layer clear.

lcdif_fb_format_t format

Frame buffer output pixel format.

lcdif_layer_convert_standard_t standard

Color space convert standard, not used for overlay layer 1 since it does not support tile YUV input.

bool enableUVSwizzle

Swizzle the U/V components, not used for overlay layer 1 since it does not support tile YUV input.

lcdif_layer_input_order_t inOrder

Color component order of the input pixel.

lcdif_layer_colorkey_t colorkey

Color key configuration.

lcdif_layer_decompress_mode_t decompress

LCDIF layer input data decompress mode.

lcdif_layer_rotate_flip_t rotateFlipMode

LCDIF frame buffer rotation or flip configuration.

lcdif_layer_alpha_blend_config_t alpha

The alpha blending configuration.

uint16_t topLeftX

The x value of thr top-left coordinate.

uint16_t topLeftY

The y value of thr top-left coordinate.

uint16_t width

The width of the layer.

uint16_t height

The height of the layer.

struct __lcdif_panel_config

#include <fsl_lcdif.h> LCDIF panel configuration.

Public Members

bool enable

Enable the frame buffer output.

bool enableGamma

Enable the gamma correction.

lcdif_layer_order_t order

Layer order.

lcdif_endian_mode_t endian

Pixel source endian mode.

struct __lcdif_cursor_config

#include <fsl_lcdif.h> LCDIF cursor configuration.

Public Members

bool enable

Enable the cursor or not.

lcdif_cursor_format_t format

Cursor format.

uint8_t hotspotOffsetX

Offset of the hotspot to top left point, range 0 ~ 31

uint8_t hotspotOffsetY

Offset of the hotspot to top left point, range 0 ~ 31

struct __lcdif_dither_config

#include <fsl_lcdif.h> LCDIF dither configuration.

a. Decide which bit of pixel color to enhance. This is configured by the lcdif_dither_config_t::redSize, lcdif_dither_config_t::greenSize, and lcdif_dither_config_t::blueSize. For example, setting redSize=6 means it is the 6th bit starting from the MSB that we want to enhance, in other words, it is the RedColor[2]bit from RedColor[7:0]. greenSize and blueSize function in the same way.

b. Create the look-up table. a. The Look-Up Table includes 16 entries, 4 bits for each. b. The Look-Up Table provides a value U[3:0] through the index X[1:0] and Y[1:0]. c. The color value RedColor[3:0] is used to compare with this U[3:0]. d. If RedColor[3:0] > U[3:0], and RedColor[7:2] is not 6'b111111, then the final color value is: NewRedColor = RedColor[7:2] + 1'b1. e. If RedColor[3:0] <= U[3:0], then NewRedColor = RedColor[7:2].

Public Members

bool enable

Enable or not.

uint32_t low

Low part of the look up table.

uint32_t high

High part of the look up table.

struct __lcdif_dbi_config

#include <fsl_lcdif.h> LCDIF DBI configuration.

Public Members

lcdif_dbi_out_swizzle_t swizzle

Swizzle.

lcdif_dbi_out_format_t format

Output format.

uint8_t acTimeUnit

Time unit for AC characteristics.

lcdif_dbi_type_t type

DBI type.

bool reversePolarity

Reverse the DC pin polarity.

uint16_t writeWRPeriod

WR signal period, Cycle number = writeWRPeriod * (acTimeUnit + 1), must be no less than 3. Only for type A and type b.

uint8_t writeWRAssert

Cycle number = writeWRAssert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Not used. With kLCDIF_DbiTypeA_ClockedE: Time to assert E. With kLCDIF_DbiTypeB: Time to assert WRX.

uint8_t writeCSAssert

Cycle number = writeCSAssert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Time to assert CSX. With kLCDIF_DbiTypeA_ClockedE: Not used. With kLCDIF_DbiTypeB: Time to assert CSX.

uint16_t writeWRDeassert

Cycle number = writeWRDeassert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Not used. With kLCDIF_DbiTypeA_ClockedE: Time to de-assert E. With kLCDIF_DbiTypeB: Time to de-assert WRX.

uint16_t writeCSDeassert

Cycle number = writeCSDeassert * (acTimeUnit + 1), only for type A and type B. With kLCDIF_DbiTypeA_FixedE: Time to de-assert CSX. With kLCDIF_DbiTypeA_ClockedE: Not used. With kLCDIF_DbiTypeB: Time to de-assert CSX.

uint8_t typeCTas

How many sdaClk cycles in Tas phase, only for Type C option 3, at least 1.

uint8_t typeCSCLTwrl

How many sdaClk cycles in Twrl phase, only for Type C, at least 1.

uint8_t typeCSCLTwrh

How many sdaClk cycles in Twrh phase, only for Type C, at least 1.

2.46 LPADC: 12-bit SAR Analog-to-Digital Converter Driver

enum _lpadc_status_flags

Define hardware flags of the module.

Values:

enumerator kLPADC_ResultFIFO0OverflowFlag

Indicates that more data has been written to the Result FIFO 0 than it can hold.

enumerator kLPADC_ResultFIFO0ReadyFlag

Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

enumerator kLPADC_TriggerExceptionFlag

Indicates that a trigger exception event has occurred.

enumerator kLPADC_TriggerCompletionFlag

Indicates that a trigger completion event has occurred.

enumerator kLPADC_CalibrationReadyFlag

Indicates that the calibration process is done.

enumerator kLPADC_ActiveFlag

Indicates that the ADC is in active state.

enumerator kLPADC_ResultFIFOOverflowFlag

To compilitable with old version, do not recommend using this, please use *kLPADC_ResultFIFO0OverflowFlag* as instead.

enumerator kLPADC_ResultFIFOReadyFlag

To compilitable with old version, do not recommend using this, please use *kLPADC_ResultFIFO0ReadyFlag* as instead.

enum _lpadc_interrupt_enable

Define interrupt switchers of the module.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_ResultFIFO0OverflowInterruptEnable

Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

enumerator kLPADC_FIFO0WatermarkInterruptEnable

Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

enumerator kLPADC_ResultFIFOOverflowInterruptEnable

To compilitable with old version, do not recommend using this, please use *kLPADC_ResultFIFO0OverflowInterruptEnable* as instead.

enumerator kLPADC_FIFOWatermarkInterruptEnable

To compilitable with old version, do not recommend using this, please use *kLPADC_FIFO0WatermarkInterruptEnable* as instead.

enumerator kLPADC_TriggerExceptionInterruptEnable

Configures ADC to generate trigger exception interrupt.

enumerator kLPADC_Trigger0CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 0 completion.

enumerator kLPADC_Trigger1CompletionInterruptEnable

Configures ADC to generate interrupt when trigger 1 completion.

enum _lpadc_trigger_status_flags

The enumerator of lpadc trigger status flags, including interrupted flags and completed flags.

Note: LPADC of different chips supports different number of trigger sources, please check the Reference Manual for details.

Values:

enumerator kLPADC_Trigger0InterruptedFlag

Trigger 0 is interrupted by a high priority exception.

enumerator kLPADC_Trigger1InterruptedFlag

Trigger 1 is interrupted by a high priority exception.

enumerator kLPADC_Trigger0CompletedFlag

Trigger 0 is completed and trigger 0 has enabled completion interrupts.

enumerator kLPADC_Trigger1CompletedFlag

Trigger 1 is completed and trigger 1 has enabled completion interrupts.

enum _lpadc_sample_scale_mode

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

Values:

enumerator kLPADC_SamplePartScale

Use divided input voltage signal. (For scale select,please refer to the reference manual).

enumerator kLPADC_SampleFullScale

Full scale (Factor of 1).

enum _lpadc_sample_channel_mode

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

Values:

enumerator kLPADC_SampleChannelSingleEndSideA

Single-end mode, only A-side channel is converted.

enumerator kLPADC_SampleChannelSingleEndSideB

Single-end mode, only B-side channel is converted.

enumerator kLPADC_SampleChannelDiffBothSideAB

Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDiffBothSideBA

Differential mode, the ADC result is (CHnB-CHnA).

enumerator kLPADC_SampleChannelDiffBothSide

Differential mode, the ADC result is (CHnA-CHnB).

enumerator kLPADC_SampleChannelDualSingleEndBothSide

Dual-Single-Ended Mode. Both A side and B side channels are converted independently.

enum _lpadc_hardware_average_mode

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

Values:

enumerator kLPADC_HardwareAverageCount1

Single conversion.

enumerator kLPADC_HardwareAverageCount2

2 conversions averaged.

enumerator kLPADC_HardwareAverageCount4

4 conversions averaged.

enumerator kLPADC_HardwareAverageCount8

8 conversions averaged.

enumerator kLPADC_HardwareAverageCount16
16 conversions averaged.

enumerator kLPADC_HardwareAverageCount32
32 conversions averaged.

enumerator kLPADC_HardwareAverageCount64
64 conversions averaged.

enumerator kLPADC_HardwareAverageCount128
128 conversions averaged.

enum _lpadc_sample_time_mode
Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

Values:

enumerator kLPADC_SampleTimeADCK3
3 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK5
5 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK7
7 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK11
11 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK19
19 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK35
35 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK67
69 ADCK cycles total sample time.

enumerator kLPADC_SampleTimeADCK131
131 ADCK cycles total sample time.

enum _lpadc_hardware_compare_mode
Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally

only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

Values:

enumerator kLPADC_HardwareCompareDisabled

Compare disabled.

enumerator kLPADC_HardwareCompareStoreOnTrue

Compare enabled. Store on true.

enumerator kLPADC_HardwareCompareRepeatUntilTrue

Compare enabled. Repeat channel acquisition until true.

enum _lpadc_conversion_resolution_mode

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to *lpadc_sample_channel_mode_t*

Values:

enumerator kLPADC_ConversionResolutionStandard

Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.

enumerator kLPADC_ConversionResolutionHigh

High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

enum _lpadc_conversion_average_mode

Define enumeration of conversion averages mode.

Configure the converion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of CAL_AVGS field in CTRL register.

Values:

enumerator kLPADC_ConversionAverage1

Single conversion.

enumerator kLPADC_ConversionAverage2

2 conversions averaged.

enumerator kLPADC_ConversionAverage4

4 conversions averaged.

enumerator kLPADC_ConversionAverage8

8 conversions averaged.

enumerator kLPADC_ConversionAverage16
16 conversions averaged.

enumerator kLPADC_ConversionAverage32
32 conversions averaged.

enumerator kLPADC_ConversionAverage64
64 conversions averaged.

enumerator kLPADC_ConversionAverage128
128 conversions averaged.

enumerator kLPADC_ConversionAverageMax

enum _lpadc_reference_voltage_mode
Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

Values:

enumerator kLPADC_ReferenceVoltageAlt1
Option 1 setting.

enumerator kLPADC_ReferenceVoltageAlt2
Option 2 setting.

enumerator kLPADC_ReferenceVoltageAlt3
Option 3 setting.

enum _lpadc_power_level_mode
Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

Values:

enumerator kLPADC_PowerLevelAlt1
Lowest power setting.

enumerator kLPADC_PowerLevelAlt2
Next lowest power setting.

enumerator kLPADC_PowerLevelAlt3
...

enumerator kLPADC_PowerLevelAlt4
Highest power setting.

enum _lpadc_offset_calibration_mode

Define enumeration of offset calibration mode.

Values:

enumerator kLPADC_OffsetCalibration12bitMode

12 bit offset calibration mode.

enumerator kLPADC_OffsetCalibration16bitMode

16 bit offset calibration mode.

enum _lpadc_trigger_priority_policy

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

Note: kLPADC_TriggerPriorityPreemptSubsequently is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

Values:

enumerator kLPADC_ConvPreemptImmediatelyNotAutoResumed

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion is not automatically resumed or restarted.

enumerator kLPADC_ConvPreemptSoftlyNotAutoResumed

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion is not resumed or restarted.

enumerator kLPADC_ConvPreemptImmediatelyAutoRestarted

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator kLPADC_ConvPreemptSoftlyAutoRestarted

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will automatically be restarted.

enumerator kLPADC_ConvPreemptImmediatelyAutoResumed

If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started, when higher priority conversion finishes, the preempted conversion will automatically be resumed.

enumerator kLPADC_ConvPreemptSoftlyAutoResumed

If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated, when higher priority conversion finishes, the preempted conversion will be automatically be resumed.

enumerator kLPADC_TriggerPriorityPreemptImmediately

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator kLPADC_TriggerPriorityPreemptSoftly

Legacy support is not recommended as it only ensures compatibility with older versions.

enumerator kLPADC_TriggerPriorityExceptionDisabled

High priority trigger exception disabled.

enum _lpadc_tune_value

Define enumeration of tune value.

Values:

enumerator kLPADC_TuneValue0

Tune value 0.

enumerator kLPADC_TuneValue1

Tune value 1.

enumerator kLPADC_TuneValue2

Tune value 2.

enumerator kLPADC_TuneValue3

Tune value 3.

typedef enum *_lpadc_sample_scale_mode* lpadc_sample_scale_mode_t

Define enumeration of sample scale mode.

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

typedef enum *_lpadc_sample_channel_mode* lpadc_sample_channel_mode_t

Define enumeration of channel sample mode.

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

typedef enum *_lpadc_hardware_average_mode* lpadc_hardware_average_mode_t

Define enumeration of hardware average selection.

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Note: Some enumerator values are not available on some devices, mainly depends on the size of AVGS field in CMDH register.

---

typedef enum *_lpadc_sample_time_mode* lpadc_sample_time_mode_t

Define enumeration of sample time selection.

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

typedef enum *_lpadc_hardware_compare_mode* lpadc_hardware_compare_mode_t

Define enumeration of hardware compare mode.

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

typedef enum *_lpadc_conversion_resolution_mode* lpadc_conversion_resolution_mode_t

Define enumeration of conversion resolution mode.

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to *lpadc_sample_channel_mode_t*

typedef enum *_lpadc_conversion_average_mode* lpadc_conversion_average_mode_t

Define enumeration of conversion averages mode.

Configure the converion average number for auto-calibration.

Note: Some enumerator values are not available on some devices, mainly depends on the size of CAL_AVGS field in CTRL register.

---

typedef enum *_lpadc_reference_voltage_mode* lpadc_reference_voltage_source_t

Define enumeration of reference voltage source.

For detail information, need to check the SoC's specification.

typedef enum *_lpadc_power_level_mode* lpadc_power_level_mode_t

Define enumeration of power configuration.

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

typedef enum *_lpadc_offset_calibration_mode* lpadc_offset_calibration_mode_t

Define enumeration of offset calibration mode.

typedef enum *_lpadc_trigger_priority_policy* lpadc_trigger_priority_policy_t

Define enumeration of trigger priority policy.

This selection controls how higher priority triggers are handled.

Note: **kLPADC_TriggerPriorityPreemptSubsequently** is not available on some devices, mainly depends on the size of TPRICTRL field in CFG register.

---

typedef enum *_lpadc_tune_value* lpadc_tune_value_t

Define enumeration of tune value.

typedef struct *_lpadc_calibration_value* lpadc_calibration_value_t

A structure of calibration value.

LPADC_CONVERSION_COMPLETE_TIMEOUT

Max loops to wait for LPADC conversion complete.

When doing calibration, driver will wait for the completion of conversion. This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

LPADC_CALIBRATION_READY_TIMEOUT

Max loops to wait for LPADC calibration ready.

Before doing calibration, driver will wait for the calibration ready. This parameter defines how many loops to check the calibration ready. If defined as 0, driver will wait forever until ready.

LPADC_GAIN_CAL_READY_TIMEOUT

Max loops to wait for LPADC gain calibration GAIN_CAL ready.

Before doing calibration, driver will wait for the gain calibration GAIN_CAL ready. This parameter defines how many loops to check the gain calibration GAIN_CAL ready. If defined as 0, driver will wait forever until ready.

LPADC_USE_FIXED_POINT_GAIN_CALCULATION

Use fixed point arithmetic for the auto-calibration gain calculation.

When set to 1, *LPADC_FinishAutoCalibration()* calculates the gain conversion result with integer-only arithmetic instead of the float/double software library, which saves several kB of flash on code size sensitive devices. The fixed point result keeps the gain fraction in Q17.14, so the two least significant bits of the gain conversion result are always zero.

ADC_OFSTRIM_OFSTRIM_MAX

ADC_OFSTRIM_OFSTRIM_SIGN

LPADC_GET_ACTIVE_COMMAND_STATUS(statusVal)

Define the MACRO function to get command status from status value.

The statusVal is the return value from *LPADC_GetStatusFlags()*.

LPADC_GET_ACTIVE_TRIGGER_STATUE(statusVal)

Define the MACRO function to get trigger status from status value.

The statusVal is the return value from *LPADC_GetStatusFlags()*.

void LPADC_Init(ADC_Type *base, const *lpadc_config_t* *config)

Initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.
- config – Pointer to configuration structure. See "lpadc_config_t".

void LPADC_GetDefaultConfig(*lpadc_config_t* *config)

Gets an available pre-defined settings for initial configuration.

This function initializes the converter configuration structure with an available settings. The default values are:

```
config->enableInDozeMode        = true;
config->enableAnalogPreliminary = false;
config->powerUpDelay            = 0x80;
config->referenceVoltageSource  = kLPADC_ReferenceVoltageAlt1;
config->powerLevelMode          = kLPADC_PowerLevelAlt1;
config->triggerPriorityPolicy   = kLPADC_TriggerPriorityPreemptImmediately;
config->enableConvPause         = false;
config->convPauseDelay          = 0U;
config->FIFOWatermark           = 0U;
```

Parameters

- config – Pointer to configuration structure.

void LPADC_Deinit(ADC_Type *base)

De-initializes the LPADC module.

Parameters

- base – LPADC peripheral base address.

static inline void LPADC_Enable(ADC_Type *base, bool enable)

Switch on/off the LPADC module.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the module.

static inline void LPADC_DoResetFIFO(ADC_Type *base)

Do reset the conversion FIFO.

Parameters

- base – LPADC peripheral base address.

static inline void LPADC_DoResetConfig(ADC_Type *base)

Do reset the module's configuration.

Reset all ADC internal logic and registers, except the Control Register (ADCx_CTRL).

Parameters

- base – LPADC peripheral base address.

static inline uint32_t LPADC_GetStatusFlags(ADC_Type *base)

Get status flags.

Parameters

- base – LPADC peripheral base address.

Returns
status flags' mask. See to *_lpadc_status_flags*.

static inline void LPADC_ClearStatusFlags(ADC_Type *base, uint32_t mask)

Clear status flags.

Only the flags can be cleared by writing ADCx_STATUS register would be cleared by this API.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for flags to be cleared. See to *_lpadc_status_flags*.

static inline uint32_t LPADC_GetTriggerStatusFlags(ADC_Type *base)

Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.

Note: On some devices, the trigger completion status may be asserted before the final command in a chained trigger sequence starts to execute. When using chained commands, do not rely on trigger completion status alone to guarantee that all conversion results are already available in the FIFO. Use FIFO ready indication together with result tags, or stall the last command with WAIT_TRIG when that sequencing model is acceptable for the application.

Parameters

- base – LPADC peripheral base address.

Returns
The OR'ed value of *_lpadc_trigger_status_flags*.

static inline void LPADC_ClearTriggerStatusFlags(ADC_Type *base, uint32_t mask)

Clear trigger status flags.

Parameters

- base – LPADC peripheral base address.
- mask – The mask of trigger status flags to be cleared, should be the OR'ed value of *_lpadc_trigger_status_flags*.

static inline void LPADC_EnableInterrupts(ADC_Type *base, uint32_t mask)

Enable interrupts.

Note: When enabling trigger completion interrupts (kLPADC_TriggerXCompletionInterruptEnable) on some devices, the interrupt may occur before the final command in a chained trigger sequence starts to execute. For multi-command trigger sequences, do not use the trigger completion interrupt alone as the indication that all expected results are already stored in the FIFO.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to *_lpadc_interrupt_enable*.

static inline void LPADC_DisableInterrupts(ADC_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – LPADC peripheral base address.
- mask – Mask value for interrupt events. See to *_lpadc_interrupt_enable*.

static inline void LPADC_EnableFIFOWatermarkDMA(ADC_Type *base, bool enable)

Switch on/off the DMA trigger for FIFO watermark event.

Parameters

- base – LPADC peripheral base address.
- enable – Switcher to the event.

static inline uint32_t LPADC_GetConvResultCount(ADC_Type *base)

Get the count of result kept in conversion FIFO.

Parameters

- base – LPADC peripheral base address.

Returns

The count of result kept in conversion FIFO.

bool LPADC_GetConvResult(ADC_Type *base, *lpadc_conv_result_t* *result)

Get the result in conversion FIFO.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

Returns

Status whether FIFO entry is valid.

void LPADC_GetConvResultBlocking(ADC_Type *base, *lpadc_conv_result_t* *result)

Get the result in conversion FIFO using blocking method.

Parameters

- base – LPADC peripheral base address.
- result – Pointer to structure variable that keeps the conversion result in conversion FIFO.

void LPADC_SetConvTriggerConfig(ADC_Type *base, uint32_t triggerId, const *lpadc_conv_trigger_config_t* *config)

Configure the conversion trigger source.

Each programmable trigger can launch the conversion command in command buffer.

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- config – Pointer to configuration structure. See to *lpadc_conv_trigger_config_t*.

void LPADC_GetDefaultConvTriggerConfig(*lpadc_conv_trigger_config_t* *config)

Gets an available pre-defined settings for trigger's configuration.

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
config->targetCommandId       = 0U;
config->delayPower            = 0U;
config->priority              = 0U;
config->channelAFIFOSelect    = 0U;
config->channelBFIFOSelect    = 0U;
config->enableHardwareTrigger = false;
```

Parameters

- config – Pointer to configuration structure.

static inline void LPADC_DoSoftwareTrigger(ADC_Type *base, uint32_t triggerIdMask)

Do software trigger to conversion command.

Parameters

- base – LPADC peripheral base address.
- triggerIdMask – Mask value for software trigger indexes, which count from zero.

static inline void LPADC_EnableHardwareTriggerCommandSelection(ADC_Type *base, uint32_t triggerId, bool enable)

Enable hardware trigger command selection.

This function will use the hardware trigger command from ADC_ETC.The trigger command is then defined by ADC hardware trigger command selection field in ADC_ETC->TRIGx_CHAINy_z_n[CSEL].

Parameters

- base – LPADC peripheral base address.
- triggerId – ID for each trigger. Typically, the available value range is from 0.
- enable – True to enable or flase to disable.

void LPADC_SetConvCommandConfig(ADC_Type *base, uint32_t commandId, const *lpadc_conv_command_config_t* *config)

Configure conversion command.

Note: The number of compare value register on different chips is different, that is mean in some chips, some command buffers do not have the compare functionality.

Parameters

- base – LPADC peripheral base address.
- commandId – ID for command in command buffer. Typically, the available value range is 1 - 15.
- config – Pointer to configuration structure. See to *lpadc_conv_command_config_t*.

void LPADC_GetDefaultConvCommandConfig(*lpadc_conv_command_config_t* *config)

Gets an available pre-defined settings for conversion command's configuration.

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```
config->sampleScaleMode             = kLPADC_SampleFullScale;
config->channelBScaleMode           = kLPADC_SampleFullScale;
config->sampleChannelMode           = kLPADC_SampleChannelSingleEndSideA;
config->channelNumber               = 0U;
config->channelBNumber              = 0U;
config->chainedNextCommandNumber    = 0U;
config->enableAutoChannelIncrement = false;
config->loopCount                   = 0U;
config->hardwareAverageMode         = kLPADC_HardwareAverageCount1;
config->sampleTimeMode              = kLPADC_SampleTimeADCK3;
config->hardwareCompareMode         = kLPADC_HardwareCompareDisabled;
config->hardwareCompareValueHigh    = 0U;
config->hardwareCompareValueLow     = 0U;
config->conversionResolutionMode    = kLPADC_ConversionResolutionStandard;
config->enableWaitTrigger           = false;
config->enableChannelB              = false;
```

Parameters

- config – Pointer to configuration structure.

void LPADC_EnableCalibration(ADC_Type *base, bool enable)

Enable the calibration function.

When CALOFS is set, the ADC is configured to perform a calibration function anytime the ADC executes a conversion. Any channel selected is ignored and the value returned in the RESFIFO is a signed value between -31 and 31. -32 is not a valid and is never a returned value. Software should copy the lower 6- bits of the conversion result stored in the RESFIFO after a completed calibration conversion to the OFSTRIM field. The OFSTRIM field is used in normal operation for offset correction.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC_SetOffsetValue(ADC_Type *base, uint32_t value)

Set proper offset value to trim ADC.

To minimize the offset during normal operation, software should read the conversion result from the RESFIFO calibration operation and write the lower 6 bits to the OFSTRIM register.

Parameters

- base – LPADC peripheral base address.
- value – Setting offset value.

status_t LPADC_DoAutoCalibration(ADC_Type *base)

Do auto calibration.

Calibration function should be executed before using converter in application. It used the software trigger and a dummy conversion, get the offset and write them into the OFSTRIM register. It called some of functional API including:

- LPADC_EnableCalibration(...)
- LPADC_SetOffsetValue(...)

- LPADC_SetConvCommandConfig(...)
- LPADC_SetConvTriggerConfig(...)

Parameters

- base – LPADC peripheral base address.
- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void LPADC_SetOffsetValue(ADC_Type *base, int16_t value)

Set trim value for offset.

Note: For 16-bit conversions, each increment is 1/2 LSB resulting in a programmable offset range of -256 LSB to 255.5 LSB; For 12-bit conversions, each increment is 1/32 LSB resulting in a programmable offset range of -16 LSB to 15.96875 LSB.

Parameters

- base – LPADC peripheral base address.
- value – Offset trim value, is a 10-bit signed value between -512 and 511.

static inline void LPADC_GetOffsetValue(ADC_Type *base, int16_t *pValue)

Get trim value of offset.

Parameters

- base – LPADC peripheral base address.
- pValue – Pointer to the variable in type of int16_t to store offset value.

static inline void LPADC_EnableOffsetCalibration(ADC_Type *base, bool enable)

Enable the offset calibration function.

Parameters

- base – LPADC peripheral base address.
- enable – switcher to the calibration function.

static inline void LPADC_SetOffsetCalibrationMode(ADC_Type *base, *lpadc_offset_calibration_mode_t* mode)

Set offset calibration mode.

Parameters

- base – LPADC peripheral base address.
- mode – set offset calibration mode.see to *lpadc_offset_calibration_mode_t* .

status_t LPADC_DoOffsetCalibration(ADC_Type *base)

Do offset calibration.

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.

- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_PrepareAutoCalibration(ADC_Type *base)

Prepare auto calibration, LPADC_FinishAutoCalibration has to be called before using the LPADC. LPADC_DoAutoCalibration has been split in two API to avoid to be stuck too long in the function.

Parameters

- base – LPADC peripheral base address.

status_t LPADC_FinishAutoCalibration(ADC_Type *base)

Finish auto calibration start with LPADC_PrepareAutoCalibration.

Note: This feature is used for LPADC with CTRL[CALOFSMODE].

Parameters

- base – LPADC peripheral base address.

Return values

- kStatus_Success – Successfully configured.
- kStatus_Timeout – Timeout occurs while waiting completion.

void LPADC_GetCalibrationValue(ADC_Type *base, *lpadc_calibration_value_t* *ptrCalibrationValue)

Get calibration value into the memory which is defined by invoker.

Note: Please note the ADC will be disabled temporary.

Note: This function should be used after finish calibration.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to *lpadc_calibration_value_t* structure, this memory block should be always powered on even in low power modes.

status_t LPADC_SetCalibrationValue(ADC_Type *base, const *lpadc_calibration_value_t* *ptrCalibrationValue)

Set calibration value into ADC calibration registers.

Note: Please note the ADC will be disabled temporary.

Parameters

- base – LPADC peripheral base address.
- ptrCalibrationValue – Pointer to *lpadc_calibration_value_t* structure which contains ADC's calibration value.

Return values

- kStatus_Success – Successfully configured.

- kStatus_Timeout – Timeout occurs while waiting completion.

static inline void LPADC_RequestHighSpeedModeTrim(ADC_Type *base)

Request high speed mode trim calculation.

Parameters

- base – LPADC peripheral base address.

static inline int8_t LPADC_GetHighSpeedTrimValue(ADC_Type *base)

Get high speed mode trim value, the result is a 5-bit signed value between -16 and 15.

Note: The high speed mode trim value is used to minimize offset for high speed conversion.

Parameters

- base – LPADC peripheral base address.

Returns

The calculated high speed mode trim value.

static inline void LPADC_SetHighSpeedTrimValue(ADC_Type *base, int8_t trimValue)

Set high speed mode trim value.

Note: If is possible to set the trim value manually, but it is recommended to use the LPADC_RequestHighSpeedModeTrim.

Parameters

- base – LPADC peripheral base address.
- trimValue – The trim value to be set.

static inline void LPADC_EnableHighSpeedConversionMode(ADC_Type *base, bool enable)

Enable/disable high speed conversion mode, if enabled conversions complete 2 or 3 ADCK cycles sooner compared to conversion cycle counts when high speed mode is disabled.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable high speed conversion mode:
  - **true** Enable high speed conversion mode;
  - **false** Disable high speed conversion mode.

static inline void LPADC_EnableExtraCycle(ADC_Type *base, bool enable)

Enable/disable an additional ADCK cycle to conversion.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable an additional ADCK cycle to conversion:
  - **true** Enable an additional ADCK cycle to conversion;
  - **false** Disable an additional ADCK cycle to conversion.

static inline void LPADC_SetTuneValue(ADC_Type *base, *lpadc_tune_value_t* tuneValue)

Set tune value which provides some variability in how many cycles are needed to complete a conversion.

Parameters

- base – LPADC peripheral base address.
- tuneValue – The tune value to be set, please refer to *lpadc_tune_value_t*.

static inline *lpadc_tune_value_t* LPADC_GetTuneValue(ADC_Type *base)

Get tune value which provides some variability in how many cycles are needed to complete a conversion.

Parameters

- base – LPADC peripheral base address.

Returns

The tune value, please refer to *lpadc_tune_value_t*.

static inline void LPADC_EnableJustifiedLeft(ADC_Type *base, bool enable)

Enable/disable left-justify format in 12-bit single-end mode.

Parameters

- base – LPADC peripheral base address.
- enable – Used to enable/disable left-justify format in 12-bit single-end mode:
  - **true** Enable left-justify format in 12-bit single-end mode;
  - **false** Disable left-justify format in 12-bit single-end mode.

FSL_LPADC_DRIVER_VERSION

LPADC driver version 2.10.3.

struct lpadc_config_t

#include <fsl_lpadc.h> LPADC global configuration.

This structure would used to keep the settings for initialization.

Public Members

bool enableInternalClock

Enables the internally generated clock source. The clock source is used in clock selection logic at the chip level and is optionally used for the ADC clock source.

bool enableVref1LowVoltage

If voltage reference option1 input is below 1.8V, it should be “true”. If voltage reference option1 input is above 1.8V, it should be “false”.

bool enableInDozeMode

Control system transition to Stop and Wait power modes while ADC is converting. When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

lpadc_conversion_average_mode_t conversionAverageMode

Auto-Calibration Averages.

bool enableAnalogPreliminary

ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).

uint32_t powerUpDelay

When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize. The startup delay count of (powerUpDelay * 4) ADCK cycles must result in a longer delay than the analog startup time.

lpadc_reference_voltage_source_t referenceVoltageSource

Selects the voltage reference high used for conversions.

lpadc_power_level_mode_t powerLevelMode

Power Configuration Selection.

lpadc_trigger_priority_policy_t triggerPriorityPolicy

Control how higher priority triggers are handled, see to lpadc_trigger_priority_policy_t.

bool enableConvPause

Enables the ADC pausing function. When enabled, a programmable delay is inserted during command execution sequencing between LOOP iterations, between commands in a sequence, and between conversions when command is executing in “Compare Until True” configuration.

uint32_t convPauseDelay

Controls the duration of pausing during command execution sequencing. The pause delay is a count of (convPauseDelay*4) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

uint32_t FIFOWatermark

FIFOWatermark is a programmable threshold setting. When the number of datawords stored in the ADC Result FIFO is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

struct lpadc_conv_command_config_t

#include <fsl_lpadc.h> Define structure to keep the configuration for conversion command.

Public Members

lpadc_sample_scale_mode_t sampleScaleMode

Sample scale mode.

lpadc_sample_scale_mode_t channelBScaleMode

Alternate channe B Scale mode.

lpadc_sample_channel_mode_t sampleChannelMode

Channel sample mode.

uint32_t channelNumber

Channel number, select the channel or channel pair.

uint32_t channelBNumber

Alternate Channel B number, select the channel.

uint32_t chainedNextCommandNumber

Selects the next command to be executed after this command completes. 1-15 is available, 0 is to terminate the chain after this command.

bool enableAutoChannelIncrement

Loop with increment: when disabled, the "loopCount" field selects the number of times the selected channel is converted consecutively; when enabled, the "loopCount" field defines how many consecutive channels are converted as part of the command execution.

uint32_t loopCount

Selects how many times this command executes before finish and transition to the next command or Idle state. Command executes LOOP+1 times. 0-15 is available.

lpadc_hardware_average_mode_t hardwareAverageMode

Hardware average selection.

lpadc_sample_time_mode_t sampleTimeMode

Sample time selection.

lpadc_hardware_compare_mode_t hardwareCompareMode

Hardware compare selection.

uint32_t hardwareCompareValueHigh

Compare Value High. The available value range is in 16-bit.

uint32_t hardwareCompareValueLow

Compare Value Low. The available value range is in 16-bit.

lpadc_conversion_resolution_mode_t conversionResolutionMode

Conversion resolution mode.

bool enableWaitTrigger

Wait for trigger assertion before execution: when disabled, this command will be automatically executed; when enabled, the active trigger must be asserted again before executing this command.

struct lpadc_conv_trigger_config_t

#include <fsl_lpadc.h> Define structure to keep the configuration for conversion trigger.

Public Members

uint32_t targetCommandId

Select the command from command buffer to execute upon detect of the associated trigger event.

uint32_t delayPower

Select the trigger delay duration to wait at the start of servicing a trigger event. When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is 2^delayPower ADCK cycles. The available value range is 4-bit.

uint32_t priority

Sets the priority of the associated trigger source. If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

bool enableHardwareTrigger

Enable hardware trigger source to initiate conversion on the rising edge of the input trigger source or not. THe software trigger is always available.

struct lpadc_conv_result_t

#include <fsl_lpadc.h> Define the structure to keep the conversion result.

Public Members

uint32_t commandIdSource

Indicate the command buffer being executed that generated this result.

uint32_t loopCountIndex

Indicate the loop count value during command execution that generated this result.

uint32_t triggerIdSource

Indicate the trigger source that initiated a conversion and generated this result.

uint16_t convValue

Data result.

struct __lpadc_calibration_value

#include <fsl_lpadc.h> A structure of calibration value.

2.47 Lpc_freqme

void FREQME_Init(FREQME_Type *base, const *freq_measure_config_t* *config)

Initialize freqme module, set operate mode, operate mode attribute and initialize measurement cycle.

Parameters

- base – FREQME peripheral base address.
- config – The pointer to module basic configuration, please refer to *freq_measure_config_t*.

void FREQME_GetDefaultConfig(*freq_measure_config_t* *config)

Get default configuration.

```
config->operateMode = kFREQME_FreqMeasurementMode;
config->operateModeAttribute.refClkScaleFactor = 0U;
config->enableContinuousMode                   = false;
config->startMeasurement                       = false;
```

Parameters

- config – The pointer to module basic configuration, please refer to *freq_measure_config_t*.

static inline void FREQME_StartMeasurementCycle(FREQME_Type *base)

Start frequency or pulse width measurement process.

Parameters

- base – FREQME peripheral base address.

static inline void FREQME_TerminateMeasurementCycle(FREQME_Type *base)

Force the termination of any measurement cycle currently in progress and resets RESULT or just reset RESULT if the module in idle state.

Parameters

- base – FREQME peripheral base address.

static inline void FREQME_EnableContinuousMode(FREQME_Type *base, bool enable)

Enable/disable Continuous mode.

Parameters

- base – FREQME peripheral base address.
- enable – Used to enable/disable continuous mode,
  - **true** Enable Continuous mode.
  - **false** Disable Continuous mode.

static inline bool FREQME_CheckContinuousMode(FREQME_Type *base)

Check whether continuous mode is enabled.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Continuous mode is enabled, the measurement is performed continuously.
- False – Continuous mode is disabled.

static inline void FREQME_SetOperateMode(FREQME_Type *base, *freqme_operate_mode_t* operateMode)

Set operate mode of freqme module.

Parameters

- base – FREQME peripheral base address.

- operateMode – The operate mode to be set, please refer to *freqme_operate_mode_t*.

static inline bool FREQME_CheckOperateMode(FREQME_Type *base)

Check module's operate mode.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Pulse width measurement mode.
- False – Frequency measurement mode.

static inline void FREQME_SetMinExpectedValue(FREQME_Type *base, uint32_t minValue)

Set the minimum expected value for the measurement result.

Parameters

- base – FREQME peripheral base address.
- minValue – The minimum value to set, please note that this value is 31 bits width.

static inline void FREQME_SetMaxExpectedValue(FREQME_Type *base, uint32_t maxValue)

Set the maximum expected value for the measurement result.

Parameters

- base – FREQME peripheral base address.
- maxValue – The maximum value to set, please note that this value is 31 bits width.

static inline void FREQME_SetReferenceClk(FREQME_Type *base, uint32_t referenceClk)

Set the reference clock source.

Parameters

- base – FREQME peripheral base address.
- referenceClk – The reference clock source, see freqme_reference_clock_t in <device>_COMMON.h.

static inline void FREQME_SetTargetClk(FREQME_Type *base, uint32_t targetClk)

Set the target clock source.

Parameters

- base – FREQME peripheral base address.
- targetClk – The target clock source, see freqme_target_clock_t in <device>_COMMON.h.

uint32_t FREQME_CalculateTargetClkFreq(FREQME_Type *base, uint32_t refClkFrequency)

Calculate the frequency of selected target clock.

Note: The formula: Ftarget = (RESULT - 2) * Freference / 2 ^ REF_SCALE or Ftarget = (RESULT + 1) * Freference / 2 ^ REF_SCALE

Note: This function only useful when the operate mode is selected as frequency measurement mode.

Parameters

- base – FREQME peripheral base address.
- refClkFrequency – The frequency of reference clock.

Returns

The frequency of target clock, if the output result is 0, please check the module's operate mode.

static inline uint8_t FREQME_GetReferenceClkScaleValue(FREQME_Type *base)

Get reference clock scaling factor.

Parameters

- base – FREQME peripheral base address.

Returns

Reference clock scaling factor, the reference count cycle is 2 ^ ref_scale.

static inline void FREQME_SetPulsePolarity(FREQME_Type *base, *freqme_pulse_polarity_t* pulsePolarity)

Set pulse polarity when operate mode is selected as Pulse Width Measurement mode.

Parameters

- base – FREQME peripheral base address.
- pulsePolarity – The pulse polarity to be set, please refer to *freqme_pulse_polarity_t*.

static inline bool FREQME_CheckPulsePolarity(FREQME_Type *base)

Check pulse polarity when the operate mode is selected as pulse width measurement mode.

Parameters

- base – FREQME peripheral base address.

Return values

- True – Low period.
- False – High period.

static inline uint32_t FREQME_GetMeasurementResult(FREQME_Type *base)

Get measurement result.

Parameters

- base – FREQME peripheral base address.

Returns

Measurement result.

static inline uint32_t FREQME_GetInterruptStatusFlags(FREQME_Type *base)

Get interrupt status flags, such as overflow interrupt status flag, underflow interrupt status flag, and so on.

Parameters

- base – FREQME peripheral base address.

Returns

Current interrupt status flags, should be the OR'ed value of *_freqme_interrupt_status_flags*.

static inline void FREQME_ClearInterruptStatusFlags(FREQME_Type *base, uint32_t statusFlags)

Clear interrupt status flags.

Parameters

- base – FREQME peripheral base address.
- statusFlags – The combination of interrupt status flags to clear, should be the OR'ed value of *_freqme_interrupt_status_flags*.

static inline void FREQME_EnableInterrupts(FREQME_Type *base, uint32_t masks)

Enable interrupts, such as result ready interrupt, overflow interrupt and so on.

Parameters

- base – FREQME peripheral base address.
- masks – The mask of interrupts to enable, should be the OR'ed value of *_freqme_interrupt_enable*.

static inline void FREQME_DisableInterrupts(FREQME_Type *base, uint32_t masks)

Disable interrupts, such as result ready interrupt, overflow interrupt and so on.

Parameters

- base – FREQME peripheral base address.
- masks – The mask of interrupts to disable, should be the OR'ed value of *_freqme_interrupt_enable*.

FSL_FREQME_DRIVER_VERSION

FREQME driver version 2.2.0.

enum _freqme_interrupt_status_flags

The enumeration of interrupt status flags. .

Values:

enumerator kFREQME_UnderflowInterruptStatusFlag

Indicate the measurement is just done and the result is less than minimum value.

enumerator kFREQME_OverflowInterruptStatusFlag

Indicate the measurement is just done and the result is greater than maximum value.

enumerator kFREQME_ReadyInterruptStatusFlag

Indicate the measurement is just done and the result is ready to read.

enumerator kFREQME_AllInterruptStatusFlags

All interrupt status flags.

enum _freqme_interrupt_enable

The enumeration of interrupts, including underflow interrupt, overflow interrupt, and result ready interrupt. .

Values:

enumerator kFREQME_UnderflowInterruptEnable

Enable interrupt when the result is less than minimum value.

enumerator kFREQME_OverflowInterruptEnable

Enable interrupt when the result is greater than maximum value.

enumerator kFREQME_ReadyInterruptEnable

Enable interrupt when a measurement completes and the result is ready.

enum _freqme_operate_mode

FREQME module operate mode enumeration, including frequency measurement mode and pulse width measurement mode.

Values:

enumerator kFREQME_FreqMeasurementMode

The module works in the frequency measurement mode.

enumerator kFREOME_PulseWidthMeasurementMode

The module works in the pulse width measurement mode.

enum _freqme_pulse_polarity

The enumeration of pulse polarity.

Values:

enumerator kFREQME_PulseHighPeriod

Select high period of the reference clock.

enumerator kFREQME_PulseLowPeriod

Select low period of the reference clock.

typedef enum *_freqme_operate_mode* freqme_operate_mode_t

FREQME module operate mode enumeration, including frequency measurement mode and pulse width measurement mode.

typedef enum *_freqme_pulse_polarity* freqme_pulse_polarity_t

The enumeration of pulse polarity.

typedef union *_freqme_mode_attribute* freqme_mode_attribute_t

The union of operate mode attribute.

Note: If the operate mode is selected as frequency measurement mode the member **refClkScaleFactor** should be used, if the operate mode is selected as pulse width measurement mode the member **pulsePolarity** should be used.

---

typedef struct *_freq_measure_config* freq_measure_config_t

The structure of freqme module basic configuration, including operate mode, operate mode attribute and so on.

union _freqme_mode_attribute

#include <fsl_freqme.h> The union of operate mode attribute.

Note: If the operate mode is selected as frequency measurement mode the member **refClkScaleFactor** should be used, if the operate mode is selected as pulse width measurement mode the member **pulsePolarity** should be used.

Public Members

uint8_t refClkScaleFactor

Only useful in frequency measurement operate mode, used to set the reference clock counter scaling factor.

freqme_pulse_polarity_t pulsePolarity

Only Useful in pulse width measurement operate mode, used to set period polarity.

struct __freq__measure__config

#include <fsl_freqme.h> The structure of freqme module basic configuration, including operate mode, operate mode attribute and so on.

Public Members

freqme_operate_mode_t operateMode

Select operate mode, please refer to *freqme_operate_mode_t*.

freqme_mode_attribute_t operateModeAttribute

Used to set the attribute of the selected operate mode, if the operate mode is selected as *kFREQME_FreqMeasurementMode* set *freqme_mode_attribute_t::refClkScaleFactor*, if operate mode is selected as *kFREOME_PulseWidthMeasurementMode*, please set *freqme_mode_attribute_t::pulsePolarity*.

bool enableContinuousMode

Enable/disable continuous mode, if continuous mode is enable, the measurement is performed continuously and the result for the last completed measurement is available in the result register.

2.48 LPFLEXCOMM: LPFLEXCOMM Driver

2.49 LPFLEXCOMM Driver

FSL__LP__FLEXCOMM__DRIVER__VERSION

FlexCOMM driver version.

enum LP__FLEXCOMM__PERIPH__T

LP_FLEXCOMM peripheral modes.

Values:

enumerator LP_FLEXCOMM_PERIPH_NONE
No peripheral

enumerator LP_FLEXCOMM_PERIPH_LPUART
LPUART peripheral

enumerator LP_FLEXCOMM_PERIPH_LPSPI
LPSPI Peripheral

enumerator LP_FLEXCOMM_PERIPH_LPI2C
LPI2C Peripheral

enumerator LP_FLEXCOMM_PERIPH_LPI2CAndLPUART
LPI2C and LPUART Peripheral

enum _lpflexcomm_interrupt_flag
LP_FLEXCOMM interrupt source flags.
Values:

enumerator kLPFLEXCOMM_I2cSlaveInterruptFlag

enumerator kLPFLEXCOMM_I2cMasterInterruptFlag

enumerator kLPFLEXCOMM_SpiInterruptFlag

enumerator kLPFLEXCOMM_UartRxInterruptFlag

enumerator kLPFLEXCOMM_UartTxInterruptFlag

enumerator kLPFLEXCOMM_AllInterruptFlag

typedef void (*lpflexcomm_irq_handler_t)(uint32_t instance, void *handle)
Typedef for interrupt handler.

IRQn_Type const kFlexcommIrqs[]
Array with IRQ number for each LP_FLEXCOMM module.

uint32_t LP_FLEXCOMM_GetInstance(void *base)
Returns instance number for LP_FLEXCOMM module with given base address.

uint32_t LP_FLEXCOMM_GetBaseAddress(uint32_t instance)
Returns for LP_FLEXCOMM base address.

uint32_t LP_FLEXCOMM_GetInterruptStatus(uint32_t instance)
brief Returns for LP_FLEXCOMM interrupt source,see *_lpflexcomm_interrupt_flag*.

status_t LP_FLEXCOMM_Init(uint32_t instance, *LP_FLEXCOMM_PERIPH_T* periph)
Initializes LP_FLEXCOMM and selects peripheral mode according to the second parameter.

void LP_FLEXCOMM_Deinit(uint32_t instance)
Deinitializes LP_FLEXCOMM.

void LP_FLEXCOMM_SetIRQHandler(uint32_t instance, *lpflexcomm_irq_handler_t* handler, void *lpflexcommHandle, *LP_FLEXCOMM_PERIPH_T* periph)

Sets IRQ handler for given LP_FLEXCOMM module. It is used by drivers register IRQ handler according to LP_FLEXCOMM mode.

2.50 LPI2C: Low Power Inter-Integrated Circuit Driver

FSL_LPI2C_DRIVER_VERSION

LPI2C driver version.

LPI2C status return codes.

Values:

enumerator kStatus_LPI2C_Busy

The master is already performing a transfer.

enumerator kStatus_LPI2C_Idle

The slave driver is idle.

enumerator kStatus_LPI2C_Nak

The slave device sent a NAK in response to a byte.

enumerator kStatus_LPI2C_FifoError

FIFO under run or overrun.

enumerator kStatus_LPI2C_BitError

Transferred bit was not seen on the bus.

enumerator kStatus_LPI2C_ArbitrationLost

Arbitration lost error.

enumerator kStatus_LPI2C_PinLowTimeout

SCL or SDA were held low longer than the timeout.

enumerator kStatus_LPI2C_NoTransferInProgress

Attempt to abort a transfer when one is not in progress.

enumerator kStatus_LPI2C_DmaRequestFail

DMA request failed.

enumerator kStatus_LPI2C_Timeout

Timeout polling status flags.

I2C_RETRY_TIMES

Retry times for waiting flag.

2.51 LPI2C Master Driver

void LPI2C_MasterGetDefaultConfig(*lpi2c_master_config_t* *masterConfig)

Provides a default configuration for the LPI2C master peripheral.

This function provides the following default configuration for the LPI2C master peripheral:

```
masterConfig->enableMaster            = true;
masterConfig->debugEnable             = false;
masterConfig->ignoreAck               = false;
masterConfig->pinConfig               = kLPI2C_2PinOpenDrain;
masterConfig->baudRate_Hz             = 100000U;
masterConfig->busIdleTimeout_ns       = 0;
masterConfig->pinLowTimeout_ns        = 0;
masterConfig->sdaGlitchFilterWidth_ns = 0;
masterConfig->sclGlitchFilterWidth_ns = 0;
masterConfig->hostRequest.enable      = false;
masterConfig->hostRequest.source      = kLPI2C_HostRequestExternalPin;
masterConfig->hostRequest.polarity    = kLPI2C_HostRequestPinActiveHigh;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with *LPI2C_MasterInit()*.

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to *lpi2c_master_config_t*.

void LPI2C_MasterInit(LPI2C_Type *base, const *lpi2c_master_config_t* *masterConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C master peripheral.

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The LPI2C peripheral base address.
- masterConfig – User provided peripheral configuration. Use *LPI2C_MasterGetDefaultConfig()* to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void LPI2C_MasterDeinit(LPI2C_Type *base)

Deinitializes the LPI2C master peripheral.

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

void LPI2C_MasterConfigureDataMatch(LPI2C_Type *base, const *lpi2c_data_match_config_t* *matchConfig)

Configures LPI2C master data match feature.

Parameters

- base – The LPI2C peripheral base address.
- matchConfig – Settings for the data match feature.

status_t LPI2C_MasterCheckAndClearError(LPI2C_Type *base, uint32_t status)

status_t LPI2C_CheckForBusyBus(LPI2C_Type *base)

static inline void LPI2C_MasterReset(LPI2C_Type *base)

Performs a software reset.

Restores the LPI2C master peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_MasterEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as master.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as master.

static inline uint32_t LPI2C_MasterGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C master status flags.

A bit mask with the state of all LPI2C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_master_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_MasterClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C master status flag state.

The following status register flags can be cleared:

- *kLPI2C_MasterEndOfPacketFlag*
- *kLPI2C_MasterStopDetectFlag*
- *kLPI2C_MasterNackDetectFlag*
- *kLPI2C_MasterArbitrationLostFlag*
- *kLPI2C_MasterFifoErrFlag*
- *kLPI2C_MasterPinLowTimeoutFlag*
- *kLPI2C_MasterDataMatchFlag*

Attempts to clear other flags has no effect.

See also:

_lpi2c_master_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _lpi2c_master_flags enumerators OR'd together. You may pass the result of a previous call to *LPI2C_MasterGetStatusFlags()*.

static inline void LPI2C_MasterEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C master interrupt requests.

All flags except *kLPI2C_MasterBusyFlag* and *kLPI2C_MasterBusBusyFlag* can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_MasterDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C master interrupt requests.

All flags except *kLPI2C_MasterBusyFlag* and *kLPI2C_MasterBusBusyFlag* can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See _lpi2c_master_flags for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_MasterGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C master interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of _lpi2c_master_flags enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_MasterEnableDMA(LPI2C_Type *base, bool enableTx, bool enableRx)

Enables or disables LPI2C master DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableTx – Enable flag for transmit DMA request. Pass true for enable, false for disable.
- enableRx – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t LPI2C_MasterGetTxFifoAddress(LPI2C_Type *base)

Gets LPI2C master transmit data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Transmit Data Register address.

static inline uint32_t LPI2C_MasterGetRxFifoAddress(LPI2C_Type *base)

Gets LPI2C master receive data register address for DMA transfer.

Parameters

- base – The LPI2C peripheral base address.

Returns

The LPI2C Master Receive Data Register address.

static inline void LPI2C_MasterSetWatermarks(LPI2C_Type *base, size_t txWords, size_t rxWords)

Sets the watermarks for LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txWords – Transmit FIFO watermark value in words. The *kLPI2C_MasterTxReadyFlag* flag is set whenever the number of words in the transmit FIFO is equal or less than *txWords*. Writing a value equal or greater than the FIFO size is truncated.
- rxWords – Receive FIFO watermark value in words. The *kLPI2C_MasterRxReadyFlag* flag is set whenever the number of words in the receive FIFO is greater than *rxWords*. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPI2C_MasterGetFifoCounts(LPI2C_Type *base, size_t *rxCount, size_t *txCount)

Gets the current number of words in the LPI2C master FIFOs.

Parameters

- base – The LPI2C peripheral base address.
- txCount – **[out]** Pointer through which the current number of words in the transmit FIFO is returned. Pass NULL if this value is not required.
- rxCount – **[out]** Pointer through which the current number of words in the receive FIFO is returned. Pass NULL if this value is not required.

void LPI2C_MasterSetBaudRate(LPI2C_Type *base, uint32_t sourceClock_Hz, uint32_t baudRate_Hz)

Sets the I2C bus frequency for master transactions.

The LPI2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Note: Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

Parameters

- base – The LPI2C peripheral base address.
- sourceClock_Hz – LPI2C functional clock frequency in Hertz.
- baudRate_Hz – Requested bus frequency in Hertz.

static inline bool LPI2C_MasterGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t LPI2C_MasterStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a START signal and slave address on the I2C bus.

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either *kLPI2C_Read* or *kLPI2C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

static inline *status_t* LPI2C_MasterRepeatedStart(LPI2C_Type *base, uint8_t address, *lpi2c_direction_t* dir)

Sends a repeated START signal and slave address on the I2C bus.

This function is used to send a Repeated START signal when a transfer is already in progress. Like *LPI2C_MasterStart()*, it also sends the specified 7-bit address.

Note: This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

- base – The LPI2C peripheral base address.
- address – 7-bit slave device address, in bits [6:0].
- dir – Master transfer direction, either *kLPI2C_Read* or *kLPI2C_Write*. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

- kStatus_Success – Repeated START signal and address were successfully enqueued in the transmit FIFO.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.

status_t LPI2C_MasterSend(LPI2C_Type *base, void *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns *kStatus_LPI2C_Nak*.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or over run.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterStop(LPI2C_Type *base)

Sends a STOP signal on the I2C bus.

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

- base – The LPI2C peripheral base address.

Return values

- kStatus_Success – The STOP signal was successfully sent on the bus and the transaction terminated.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.

- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

status_t LPI2C_MasterTransferBlocking(LPI2C_Type *base, *lpi2c_master_transfer_t* *transfer)

Performs a master polling transfer on the I2C bus.

Note: The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

- base – The LPI2C peripheral base address.
- transfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data was received successfully.
- kStatus_LPI2C_Busy – Another master is currently utilizing the bus.
- kStatus_LPI2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_LPI2C_FifoError – FIFO under run or overrun.
- kStatus_LPI2C_ArbitrationLost – Arbitration lost error.
- kStatus_LPI2C_PinLowTimeout – SCL or SDA were held low longer than the timeout.

void LPI2C_MasterTransferCreateHandle(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C master non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the *LPI2C_MasterTransferAbort()* API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferNonBlocking(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.

- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t LPI2C_MasterTransferGetCount(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferAbort(LPI2C_Type *base, *lpi2c_master_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_LPI2C_Idle – There is not a non-blocking transaction currently in progress.

void LPI2C_MasterTransferHandleIRQ(uint32_t instance, void *lpi2cMasterHandle)

Reusable routine to handle master interrupts.

Note: This function does not need to be called unless you are reimplementing the non-blocking API's interrupt handler routines to add special functionality.

Parameters

- instance – The LPI2C instance.
- lpi2cMasterHandle – Pointer to the LPI2C master driver handle.

enum _lpi2c_master_flags

LPI2C master peripheral flags.

The following status register flags can be cleared:

- *kLPI2C_MasterEndOfPacketFlag*
- *kLPI2C_MasterStopDetectFlag*
- *kLPI2C_MasterNackDetectFlag*
- *kLPI2C_MasterArbitrationLostFlag*
- *kLPI2C_MasterFifoErrFlag*
- *kLPI2C_MasterPinLowTimeoutFlag*
- *kLPI2C_MasterDataMatchFlag*

All flags except *kLPI2C_MasterBusyFlag* and *kLPI2C_MasterBusBusyFlag* can be enabled as interrupts.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_MasterTxReadyFlag

Transmit data flag

enumerator kLPI2C_MasterRxReadyFlag

Receive data flag

enumerator kLPI2C_MasterEndOfPacketFlag

End Packet flag

enumerator kLPI2C_MasterStopDetectFlag

Stop detect flag

enumerator kLPI2C_MasterNackDetectFlag

NACK detect flag

enumerator kLPI2C_MasterArbitrationLostFlag

Arbitration lost flag

enumerator kLPI2C_MasterFifoErrFlag

FIFO error flag

enumerator kLPI2C_MasterPinLowTimeoutFlag

Pin low timeout flag

enumerator kLPI2C_MasterDataMatchFlag

Data match flag

enumerator kLPI2C_MasterBusyFlag

Master busy flag

enumerator kLPI2C_MasterBusBusyFlag
Bus busy flag

enumerator kLPI2C_MasterClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_MasterIrqFlags
IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_MasterErrorFlags
Errors to check for.

enum _lpi2c_direction
Direction of master and slave transfers.
Values:

enumerator kLPI2C_Write
Master transmit.

enumerator kLPI2C_Read
Master receive.

enum _lpi2c_master_pin_config
LPI2C pin configuration.
Values:

enumerator kLPI2C_2PinOpenDrain
LPI2C Configured for 2-pin open drain mode

enumerator kLPI2C_2PinOutputOnly
LPI2C Configured for 2-pin output only mode (ultra-fast mode)

enumerator kLPI2C_2PinPushPull
LPI2C Configured for 2-pin push-pull mode

enumerator kLPI2C_4PinPushPull
LPI2C Configured for 4-pin push-pull mode

enumerator kLPI2C_2PinOpenDrainWithSeparateSlave
LPI2C Configured for 2-pin open drain mode with separate LPI2C slave

enumerator kLPI2C_2PinOutputOnlyWithSeparateSlave
LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave

enumerator kLPI2C_2PinPushPullWithSeparateSlave
LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave

enumerator kLPI2C_4PinPushPullWithInvertedOutput
LPI2C Configured for 4-pin push-pull mode(inverted outputs)

enum _lpi2c_host_request_source
LPI2C master host request selection.
Values:

enumerator kLPI2C_HostRequestExternalPin
Select the LPI2C_HREQ pin as the host request input

enumerator kLPI2C_HostRequestInputTrigger
Select the input trigger as the host request input

enum _lpi2c_host_request_polarity
LPI2C master host request pin polarity configuration.
Values:

enumerator kLPI2C_HostRequestPinActiveLow
Configure the LPI2C_HREQ pin active low

enumerator kLPI2C_HostRequestPinActiveHigh
Configure the LPI2C_HREQ pin active high

enum _lpi2c_data_match_config_mode
LPI2C master data match configuration modes.
Values:

enumerator kLPI2C_MatchDisabled
LPI2C Match Disabled

enumerator kLPI2C_1stWordEqualsM0OrM1
LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1

enumerator kLPI2C_AnyWordEqualsM0OrM1
LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1

enumerator kLPI2C_1stWordEqualsM0And2ndWordEqualsM1
LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1

enumerator kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1
LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1

enumerator kLPI2C_1stWordAndM1EqualsM0AndM1
LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1

enumerator kLPI2C_AnyWordAndM1EqualsM0AndM1
LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1

enum _lpi2c_master_transfer_flags

Transfer option flags.

Note: These enumerations are intended to be OR'd together to form a bit mask of options for the *_lpi2c_master_transfer::flags* field.

Values:

enumerator kLPI2C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kLPI2C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kLPI2C_TransferNoStopFlag

Don't send a stop condition.

typedef enum *_lpi2c_direction* lpi2c_direction_t

Direction of master and slave transfers.

typedef enum *_lpi2c_master_pin_config* lpi2c_master_pin_config_t

LPI2C pin configuration.

typedef enum *_lpi2c_host_request_source* lpi2c_host_request_source_t

LPI2C master host request selection.

typedef enum *_lpi2c_host_request_polarity* lpi2c_host_request_polarity_t

LPI2C master host request pin polarity configuration.

typedef struct *_lpi2c_master_config* lpi2c_master_config_t

Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the *LPI2C_MasterGetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_data_match_config_mode* lpi2c_data_match_config_mode_t

LPI2C master data match configuration modes.

typedef struct *_lpi2c_match_config* lpi2c_data_match_config_t

LPI2C master data match configuration structure.

typedef struct *_lpi2c_master_transfer* lpi2c_master_transfer_t

typedef struct *_lpi2c_master_handle* lpi2c_master_handle_t

typedef void (*lpi2c_master_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_handle_t* *handle, *status_t* completionStatus, void *userData)

Master completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to *LPI2C_MasterTransferCreateHandle()*.

Param base
The LPI2C peripheral base address.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

typedef void (*lpi2c_master_isr_t)(uint32_t instance, void *handle)

Typedef for master interrupt handler, used internally for LPI2C master interrupt and EDMA transactional APIs.

struct _lpi2c_master_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C master module.

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the *LPI2C_MasterGetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableMaster

Whether to enable master mode.

bool enableDoze

Whether master is enabled in doze mode.

bool debugEnable

Enable transfers to continue when halted in debug mode.

bool ignoreAck

Whether to ignore ACK/NACK.

lpi2c_master_pin_config_t pinConfig

The pin configuration option.

uint32_t baudRate_Hz

Desired baud rate in Hertz.

uint32_t busIdleTimeout_ns

Bus idle timeout in nanoseconds. Set to 0 to disable.

uint32_t pinLowTimeout_ns

Pin low timeout in nanoseconds. Set to 0 to disable.

uint8_t sdaGlitchFilterWidth_ns
: Width in nanoseconds of glitch filter on SDA pin. Set to 0 to disable.

uint8_t sclGlitchFilterWidth_ns
: Width in nanoseconds of glitch filter on SCL pin. Set to 0 to disable.

struct *_lpi2c_master_config* hostRequest
: Host request options.

struct __lpi2c_match_config
: *#include <fsl_lpi2c.h>* LPI2C master data match configuration structure.

Public Members

lpi2c_data_match_config_mode_t matchMode
: Data match configuration setting.

bool rxDataMatchOnly
: When set to true, received data is ignored until a successful match.

uint32_t match0
: Match value 0.

uint32_t match1
: Match value 1.

struct __lpi2c_master_transfer
: *#include <fsl_lpi2c.h>* Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the *LPI2C_MasterTransferNonBlocking()* API.

Public Members

uint32_t flags
: Bit mask of options for the transfer. See enumeration *_lpi2c_master_transfer_flags* for available options. Set to 0 or *kLPI2C_TransferDefaultFlag* for normal transfers.

uint16_t slaveAddress
: The 7-bit slave address.

lpi2c_direction_t direction
: Either *kLPI2C_Read* or *kLPI2C_Write*.

uint32_t subaddress
: Sub address. Transferred MSB first.

size_t subaddressSize

Length of sub address to send in bytes. Maximum size is 4 bytes.

void *data

Pointer to data to transfer.

size_t dataSize

Number of bytes to transfer.

struct __lpi2c_master_handle

#include <fsl_lpi2c.h> Driver handle for master non-blocking APIs.

Note: The contents of this structure are private and subject to change.

Public Members

uint8_t state

Transfer state machine current state.

uint16_t remainingBytes

Remaining byte count in current state.

uint8_t *buf

Buffer pointer for current state.

uint16_t commandBuffer[6]

LPI2C command sequence. When all 6 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word]

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

uint16_t chunkSize

Remaining byte count in current chunk.

struct hostRequest

Public Members

bool enable

Enable host request.

lpi2c_host_request_source_t source

Host request source.

lpi2c_host_request_polarity_t polarity

Host request pin polarity.

2.52 LPI2C Master DMA Driver

void LPI2C_MasterCreateEDMAHandle(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *edma_handle_t* *rxDmaHandle, *edma_handle_t* *txDmaHandle, *lpi2c_master_edma_transfer_callback_t* callback, void *userData)

Create a new handle for the LPI2C master DMA APIs.

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the *LPI2C_MasterTransferAbortEDMA()* API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C master driver handle.
- rxDmaHandle – Handle for the eDMA receive channel. Created by the user prior to calling this function.
- txDmaHandle – Handle for the eDMA transmit channel. Created by the user prior to calling this function.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_MasterTransferEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *lpi2c_master_transfer_t* *transfer)

Performs a non-blocking DMA-based transaction on the I2C bus.

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- transfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_LPI2C_Busy – Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

status_t LPI2C_MasterTransferGetCountEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – There is not a DMA transaction currently in progress.

status_t LPI2C_MasterTransferAbortEDMA(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle)

Terminates a non-blocking LPI2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to the LPI2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_LPI2C_Idle – There is not a DMA transaction currently in progress.

typedef struct *_lpi2c_master_edma_handle* lpi2c_master_edma_handle_t

typedef void (*lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base, *lpi2c_master_edma_handle_t* *handle, *status_t* completionStatus, void *userData)

Master DMA completion callback function pointer type.

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to *LPI2C_MasterCreateEDMAHandle()*.

Param base
The LPI2C peripheral base address.

Param handle
Handle associated with the completed transfer.

Param completionStatus
Either kStatus_Success or an error code describing how the transfer completed.

Param userData
Arbitrary pointer-sized value passed from the application.

struct __lpi2c_master_edma_handle

#include <fsl_lpi2c_edma.h> Driver handle for master DMA APIs.

Note: The contents of this structure are private and subject to change.

Public Members

LPI2C_Type *base

LPI2C base pointer.

bool isBusy

Transfer state machine current state.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint16_t commandBuffer[10]

LPI2C command sequence. When all 10 command words are used: Start&addr&write[1 word] + subaddr[4 words] + restart&addr&read[1 word] + receive&Size[4 words]

lpi2c_master_transfer_t transfer

Copy of the current transfer info.

lpi2c_master_edma_transfer_callback_t completionCallback

Callback function pointer.

void *userData

Application data passed to callback.

edma_handle_t *rx

Handle for receive DMA channel.

edma_handle_t *tx

Handle for transmit DMA channel.

edma_tcd_t tcds[3]

Software TCD. Three are allocated to provide enough room to align to 32-bytes.

2.53 LPI2C Slave Driver

void LPI2C_SlaveGetDefaultConfig(*lpi2c_slave_config_t* *slaveConfig)

Provides a default configuration for the LPI2C slave peripheral.

This function provides the following default configuration for the LPI2C slave peripheral:

```
slaveConfig->enableSlave                = true;
slaveConfig->address0                   = 0U;
slaveConfig->address1                   = 0U;
slaveConfig->addressMatchMode           = kLPI2C_MatchAddress0;
slaveConfig->filterDozeEnable           = true;
slaveConfig->filterEnable               = true;
slaveConfig->enableGeneralCall          = false;
slaveConfig->sclStall.enableAck         = false;
slaveConfig->sclStall.enableTx          = true;
slaveConfig->sclStall.enableRx          = true;
slaveConfig->sclStall.enableAddress     = true;
slaveConfig->ignoreAck                  = false;
slaveConfig->enableReceivedAddressRead = false;
slaveConfig->sdaGlitchFilterWidth_ns    = 0;
slaveConfig->sclGlitchFilterWidth_ns    = 0;
slaveConfig->dataValidDelay_ns          = 0;
slaveConfig->clockHoldTime_ns           = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with *LPI2C_SlaveInit()*. Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

- slaveConfig – **[out]** User provided configuration structure that is set to default values. Refer to *lpi2c_slave_config_t*.

void LPI2C_SlaveInit(LPI2C_Type *base, const *lpi2c_slave_config_t* *slaveConfig, uint32_t sourceClock_Hz)

Initializes the LPI2C slave peripheral.

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

- base – The LPI2C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use *LPI2C_SlaveGetDefaultConfig()* to get a set of defaults that you can override.
- sourceClock_Hz – Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.

void LPI2C_SlaveDeinit(LPI2C_Type *base)

Deinitializes the LPI2C slave peripheral.

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveReset(LPI2C_Type *base)

Performs a software reset of the LPI2C slave peripheral.

Parameters

- base – The LPI2C peripheral base address.

static inline void LPI2C_SlaveEnable(LPI2C_Type *base, bool enable)

Enables or disables the LPI2C module as slave.

Parameters

- base – The LPI2C peripheral base address.
- enable – Pass true to enable or false to disable the specified LPI2C as slave.

static inline uint32_t LPI2C_SlaveGetStatusFlags(LPI2C_Type *base)

Gets the LPI2C slave status flags.

A bit mask with the state of all LPI2C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_lpi2c_slave_flags

Parameters

- base – The LPI2C peripheral base address.

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

static inline void LPI2C_SlaveClearStatusFlags(LPI2C_Type *base, uint32_t statusMask)

Clears the LPI2C status flag state.

The following status register flags can be cleared:

- *kLPI2C_SlaveRepeatedStartDetectFlag*
- *kLPI2C_SlaveStopDetectFlag*
- *kLPI2C_SlaveBitErrFlag*
- *kLPI2C_SlaveFifoErrFlag*

Attempts to clear other flags has no effect.

See also:

_lpi2c_slave_flags.

Parameters

- base – The LPI2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of *_lpi2c_slave_flags* enumerators OR'd together. You may pass the result of a previous call to *LPI2C_SlaveGetStatusFlags()*.

static inline void LPI2C_SlaveEnableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Enables the LPI2C slave interrupt requests.

All flags except *kLPI2C_SlaveBusyFlag* and *kLPI2C_SlaveBusBusyFlag* can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See *_lpi2c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline void LPI2C_SlaveDisableInterrupts(LPI2C_Type *base, uint32_t interruptMask)

Disables the LPI2C slave interrupt requests.

All flags except *kLPI2C_SlaveBusyFlag* and *kLPI2C_SlaveBusBusyFlag* can be enabled as interrupts.

Parameters

- base – The LPI2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See *_lpi2c_slave_flags* for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t LPI2C_SlaveGetEnabledInterrupts(LPI2C_Type *base)

Returns the set of currently enabled LPI2C slave interrupt requests.

Parameters

- base – The LPI2C peripheral base address.

Returns

A bitmask composed of *_lpi2c_slave_flags* enumerators OR'd together to indicate the set of enabled interrupts.

static inline void LPI2C_SlaveEnableDMA(LPI2C_Type *base, bool enableAddressValid, bool enableRx, bool enableTx)

Enables or disables the LPI2C slave peripheral DMA requests.

Parameters

- base – The LPI2C peripheral base address.
- enableAddressValid – Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request.
- enableRx – Enable flag for the receive data DMA request. Pass true for enable, false for disable.
- enableTx – Enable flag for the transmit data DMA request. Pass true for enable, false for disable.

static inline bool LPI2C_SlaveGetBusIdleState(LPI2C_Type *base)

Returns whether the bus is idle.

Requires the slave mode to be enabled.

Parameters

- base – The LPI2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

static inline void LPI2C_SlaveTransmitAck(LPI2C_Type *base, bool ackOrNack)

Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.

Use this function to send an ACK or NAK when the *kLPI2C_SlaveTransmitAckFlag* is asserted. This only happens if you enable the sclStall.enableAck field of the *lpi2c_slave_config_t* configuration structure used to initialize the slave peripheral.

Parameters

- base – The LPI2C peripheral base address.
- ackOrNack – Pass true for an ACK or false for a NAK.

static inline void LPI2C_SlaveEnableAckStall(LPI2C_Type *base, bool enable)

Enables or disables ACKSTALL.

When enables ACKSTALL, software can transmit either an ACK or NAK on the I2C bus in response to a byte from the master.

Parameters

- base – The LPI2C peripheral base address.
- enable – True will enable ACKSTALL,false will disable ACKSTALL.

static inline uint32_t LPI2C_SlaveGetReceivedAddress(LPI2C_Type *base)

Returns the slave address sent by the I2C master.

This function should only be called if the *kLPI2C_SlaveAddressValidFlag* is asserted.

Parameters

- base – The LPI2C peripheral base address.

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

status_t LPI2C_SlaveSend(LPI2C_Type *base, void *txBuff, size_t txSize, size_t *actualTxSize)

Performs a polling send transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- actualTxSize – **[out]**

Returns

Error or success status returned by API.

status_t LPI2C_SlaveReceive(LPI2C_Type *base, void *rxBuff, size_t rxSize, size_t *actualRxSize)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The LPI2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- actualRxSize – **[out]**

Returns

Error or success status returned by API.

void LPI2C_SlaveTransferCreateHandle(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, *lpi2c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the LPI2C slave non-blocking APIs.

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the *LPI2C_SlaveTransferAbort()* API shall be called.

Note: The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

- base – The LPI2C peripheral base address.
- handle – **[out]** Pointer to the LPI2C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t LPI2C_SlaveTransferNonBlocking(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling I2C_SlaveInit() and *LPI2C_SlaveTransferCreateHandle()* to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to *LPI2C_SlaveTransferCreateHandle()*. The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *lpi2c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kLPI2C_SlaveTransmitEvent* and *kLPI2C_SlaveReceiveEvent* events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kLPI2C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together *lpi2c_slave_transfer_event_t* enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and *kLPI2C_SlaveAllEvents* to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_LPI2C_Busy – Slave transfers have already been started on this handle.

status_t LPI2C_SlaveTransferGetCount(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer status during a non-blocking transfer.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure.
- count – **[out]** Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress –

void LPI2C_SlaveTransferAbort(LPI2C_Type *base, *lpi2c_slave_handle_t* *handle)

Aborts the slave non-blocking transfers.

Note: This API could be called at any time to stop slave for handling the bus events.

Parameters

- base – The LPI2C peripheral base address.
- handle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

Return values

- kStatus_Success –
- kStatus_LPI2C_Idle –

void LPI2C_SlaveTransferHandleIRQ(uint32_t instance, void *lpi2cSlaveHandle)

Reusable routine to handle slave interrupts.

Note: This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

- instance – The LPI2C instance.
- lpi2cSlaveHandle – Pointer to lpi2c_slave_handle_t structure which stores the transfer state.

enum _lpi2c_slave_flags

LPI2C slave peripheral flags.

The following status register flags can be cleared:

- *kLPI2C_SlaveRepeatedStartDetectFlag*
- *kLPI2C_SlaveStopDetectFlag*
- *kLPI2C_SlaveBitErrFlag*
- *kLPI2C_SlaveFifoErrFlag*

All flags except *kLPI2C_SlaveBusyFlag* and *kLPI2C_SlaveBusBusyFlag* can be enabled as interrupts.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator kLPI2C_SlaveTxReadyFlag

Transmit data flag

enumerator kLPI2C_SlaveRxReadyFlag

Receive data flag

enumerator kLPI2C_SlaveAddressValidFlag
Address valid flag

enumerator kLPI2C_SlaveTransmitAckFlag
Transmit ACK flag

enumerator kLPI2C_SlaveRepeatedStartDetectFlag
Repeated start detect flag

enumerator kLPI2C_SlaveStopDetectFlag
Stop detect flag

enumerator kLPI2C_SlaveBitErrFlag
Bit error flag

enumerator kLPI2C_SlaveFifoErrFlag
FIFO error flag

enumerator kLPI2C_SlaveAddressMatch0Flag
Address match 0 flag

enumerator kLPI2C_SlaveAddressMatch1Flag
Address match 1 flag

enumerator kLPI2C_SlaveGeneralCallFlag
General call flag

enumerator kLPI2C_SlaveBusyFlag
Master busy flag

enumerator kLPI2C_SlaveBusBusyFlag
Bus busy flag

enumerator kLPI2C_SlaveClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kLPI2C_SlaveIrqFlags
IRQ sources enabled by the non-blocking transactional API.

enumerator kLPI2C_SlaveErrorFlags
Errors to check for.

enum __lpi2c_slave_address_match
LPI2C slave address match options.
Values:

enumerator kLPI2C_MatchAddress0
Match only address 0.

enumerator kLPI2C_MatchAddress0OrAddress1

Match either address 0 or address 1.

enumerator kLPI2C_MatchAddress0ThroughAddress1

Match a range of slave addresses from address 0 through address 1.

enum _lpi2c_slave_transfer_event

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to *LPI2C_SlaveTransferNonBlocking()* in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

Values:

enumerator kLPI2C_SlaveAddressMatchEvent

Received the slave address after a start or repeated start.

enumerator kLPI2C_SlaveTransmitEvent

Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kLPI2C_SlaveReceiveEvent

Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kLPI2C_SlaveTransmitAckEvent

Callback needs to either transmit an ACK or NACK. When this event is set, the driver will no longer decide to reply to ack/nack.

enumerator kLPI2C_SlaveRepeatedStartEvent

A repeated start was detected.

enumerator kLPI2C_SlaveCompletionEvent

A stop was detected, completing the transfer.

enumerator kLPI2C_SlaveAllEvents

Bit mask of all available events.

typedef enum *_lpi2c_slave_address_match* lpi2c_slave_address_match_t

LPI2C slave address match options.

typedef struct *_lpi2c_slave_config* lpi2c_slave_config_t

Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the *LPI2C_SlaveGetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

typedef enum *_lpi2c_slave_transfer_event* lpi2c_slave_transfer_event_t

Set of events sent to the callback for non blocking slave transfers.

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to *LPI2C_SlaveTransferNonBlocking()* in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note: These enumerations are meant to be OR'd together to form a bit mask of events.

typedef struct *_lpi2c_slave_transfer* lpi2c_slave_transfer_t

LPI2C slave transfer structure.

typedef struct *_lpi2c_slave_handle* lpi2c_slave_handle_t

typedef void (*lpi2c_slave_transfer_callback_t)(LPI2C_Type *base, *lpi2c_slave_transfer_t* *transfer, void *userData)

Slave event callback function pointer type.

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C_SlaveSetCallback() function after you have created a handle.

Param base
Base address for the LPI2C instance on which the event occurred.

Param transfer
Pointer to transfer descriptor containing values passed to and/or from the callback.

Param userData
Arbitrary pointer-sized value passed from the application.

struct _lpi2c_slave_config

#include <fsl_lpi2c.h> Structure with settings to initialize the LPI2C slave module.

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the *LPI2C_SlaveGetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableSlave

Enable slave mode.

uint8_t address0

Slave's 7-bit address.

uint8_t address1

Alternate slave 7-bit address.

lpi2c_slave_address_match_t addressMatchMode
Address matching options.

bool filterDozeEnable
Enable digital glitch filter in doze mode.

bool filterEnable
Enable digital glitch filter.

bool enableGeneralCall
Enable general call address matching.

bool ignoreAck
Continue transfers after a NACK is detected.

bool enableReceivedAddressRead
Enable reading the address received address as the first byte of data.

uint32_t sdaGlitchFilterWidth_ns
Width in nanoseconds of the digital filter on the SDA signal. Set to 0 to disable.

uint32_t sclGlitchFilterWidth_ns
Width in nanoseconds of the digital filter on the SCL signal. Set to 0 to disable.

uint32_t dataValidDelay_ns
Width in nanoseconds of the data valid delay.

uint32_t clockHoldTime_ns
Width in nanoseconds of the clock hold time.

struct _lpi2c_slave_transfer
#include <fsl_lpi2c.h> LPI2C slave transfer structure.

Public Members

lpi2c_slave_transfer_event_t event
Reason the callback is being invoked.

uint8_t receivedAddress
Matching address send by master.

uint8_t *data
Transfer buffer

size_t dataSize
Transfer size

status_t completionStatus

Success or error code describing how the transfer completed. Only applies for *kLPI2C_SlaveCompletionEvent*.

size_t transferredCount

Number of bytes actually transferred since start or last repeated start.

struct __lpi2c_slave_handle

#include <fsl_lpi2c.h> LPI2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

lpi2c_slave_transfer_t transfer

LPI2C slave transfer copy.

bool isBusy

Whether transfer is busy.

bool wasTransmit

Whether the last transfer was a transmit.

uint32_t eventMask

Mask of enabled events.

uint32_t transferredCount

Count of bytes transferred.

lpi2c_slave_transfer_callback_t callback

Callback function called at transfer event.

void *userData

Callback parameter passed to callback.

struct sclStall

Public Members

bool enableAck

Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted. Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDataSCLStall or enableAddressSCLStall.

bool enableTx

Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.

bool enableRx

Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.

bool enableAddress

Enables SCL clock stretching when the address valid flag is asserted.

2.54 LPSPI: Low Power Serial Peripheral Interface

2.55 LPSPI Peripheral driver

void LPSPI_MasterInit(LPSPI_Type *base, const *lpspi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Initializes the LPSPI master.

Parameters

- base – LPSPI peripheral address.
- masterConfig – Pointer to structure lpspi_master_config_t.
- srcClock_Hz – Module source input clock in Hertz

void LPSPI_MasterGetDefaultConfig(*lpspi_master_config_t* *masterConfig)

Sets the lpspi_master_config_t structure to default values.

This API initializes the configuration structure for *LPSPI_MasterInit()*. The initialized structure can remain unchanged in *LPSPI_MasterInit()*, or can be modified before calling the *LPSPI_MasterInit()*. Example:

```
lpspi_master_config_t masterConfig;
LPSPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – pointer to lpspi_master_config_t structure

void LPSPI_SlaveInit(LPSPI_Type *base, const *lpspi_slave_config_t* *slaveConfig)

LPSPI slave configuration.

Parameters

- base – LPSPI peripheral address.
- slaveConfig – Pointer to a structure lpspi_slave_config_t.

void LPSPI_SlaveGetDefaultConfig(*lpspi_slave_config_t* *slaveConfig)

Sets the lpspi_slave_config_t structure to default values.

This API initializes the configuration structure for *LPSPI_SlaveInit()*. The initialized structure can remain unchanged in *LPSPI_SlaveInit()* or can be modified before calling the *LPSPI_SlaveInit()*. Example:

```
lpspi_slave_config_t slaveConfig;
LPSPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – pointer to lpspi_slave_config_t structure.

void LPSPI_Deinit(LPSPI_Type *base)

De-initializes the LPSPI peripheral. Call this API to disable the LPSPI clock.

Parameters

- base – LPSPI peripheral address.

void LPSPI_Reset(LPSPI_Type *base)

Restores the LPSPI peripheral to reset state. Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

Parameters

- base – LPSPI peripheral address.

uint32_t LPSPI_GetInstance(LPSPI_Type *base)

Get the LPSPI instance from peripheral base address.

Parameters

- base – LPSPI peripheral base address.

Returns

LPSPI instance.

static inline void LPSPI_Enable(LPSPI_Type *base, bool enable)

Enables the LPSPI peripheral and sets the MCR MDIS to 0.

Parameters

- base – LPSPI peripheral address.
- enable – Pass true to enable module, false to disable module.

static inline uint32_t LPSPI_GetStatusFlags(LPSPI_Type *base)

Gets the LPSPI status flag state.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI status(in SR register).

static inline uint8_t LPSPI_GetTxFifoSize(LPSPI_Type *base)

Gets the LPSPI Tx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Tx FIFO size.

static inline uint8_t LPSPI_GetRxFifoSize(LPSPI_Type *base)

Gets the LPSPI Rx FIFO size.

Parameters

- base – LPSPI peripheral address.

Returns
The LPSPI Rx FIFO size.

static inline uint32_t LPSPI_GetTxFifoCount(LPSPI_Type *base)

Gets the LPSPI Tx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns
The number of words in the transmit FIFO.

static inline uint32_t LPSPI_GetRxFifoCount(LPSPI_Type *base)

Gets the LPSPI Rx FIFO count.

Parameters

- base – LPSPI peripheral address.

Returns
The number of words in the receive FIFO.

static inline void LPSPI_ClearStatusFlags(LPSPI_Type *base, uint32_t statusFlags)

Clears the LPSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the _lpspi_flags. Example usage:

```
LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|kLPSPI_RxDataReadyFlag);
```

Parameters

- base – LPSPI peripheral address.
- statusFlags – The status flag used from type _lpspi_flags.

static inline uint32_t LPSPI_GetTcr(LPSPI_Type *base)

static inline void LPSPI_EnableInterrupts(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI interrupts.

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_interrupt_enable.

static inline void LPSPI_DisableInterrupts(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI interrupts.

```
LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable | kLPSPI_RxInterruptEnable );
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_interrupt_enable.

static inline void LPSPI_EnableDMA(LPSPI_Type *base, uint32_t mask)

Enables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_dma_enable.

static inline void LPSPI_DisableDMA(LPSPI_Type *base, uint32_t mask)

Disables the LPSPI DMA request.

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
SPI_DisableDMA(base, kLPSPI_TxDmaEnable | kLPSPI_RxDmaEnable);
```

Parameters

- base – LPSPI peripheral address.
- mask – The interrupt mask; Use the enum _lpspi_dma_enable.

static inline uint32_t LPSPI_GetTxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Transmit Data Register address for a DMA operation.

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Transmit Data Register address.

static inline uint32_t LPSPI_GetRxRegisterAddress(LPSPI_Type *base)

Gets the LPSPI Receive Data Register address for a DMA operation.

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns

The LPSPI Receive Data Register address.

bool LPSPI_CheckTransferArgument(LPSPI_Type *base, *lpspi_transfer_t* *transfer, bool isEdma)

Check the argument for transfer .

Parameters

- base – LPSPI peripheral address.
- transfer – the transfer struct to be used.
- isEdma – True to check for EDMA transfer, false to check interrupt non-blocking transfer

Returns
Return true for right and false for wrong.

static inline void LPSPI_SetMasterSlaveMode(LPSPI_Type *base, *lpspi_master_slave_mode_t* mode)

Configures the LPSPI for either master or slave.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

Parameters

- base – LPSPI peripheral address.
- mode – Mode setting (master or slave) of type lpspi_master_slave_mode_t.

static inline void LPSPI_SelectTransferPCS(LPSPI_Type *base, *lpspi_which_pcs_t* select)

Configures the peripheral chip select used for the transfer.

Parameters

- base – LPSPI peripheral address.
- select – LPSPI Peripheral Chip Select (PCS) configuration.

static inline void LPSPI_SetPCSContinous(LPSPI_Type *base, bool IsContinous)

Set the PCS signal to continuous or uncontinuous mode.

Note: In master mode, continuous transfer will keep the PCS asserted at the end of the frame size, until a command word is received that starts a new frame. So PCS must be set back to uncontinuous when transfer finishes. In slave mode, when continuous transfer is enabled, the LPSPI will only transmit the first frame size bits, after that the LPSPI will transmit received data back (assuming a 32-bit shift register).

Parameters

- base – LPSPI peripheral address.
- IsContinous – True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

static inline bool LPSPI_IsMaster(LPSPI_Type *base)

Returns whether the LPSPI module is in master mode.

Parameters

- base – LPSPI peripheral address.

Returns
Returns true if the module is in master mode or false if the module is in slave mode.

static inline void LPSPI_FlushFifo(LPSPI_Type *base, bool flushTxFifo, bool flushRxFifo)

Flushes the LPSPI FIFOs.

Parameters

- base – LPSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO.
- flushRxFifo – Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO.

static inline void LPSPI_SetFifoWatermarks(LPSPI_Type *base, uint32_t txWater, uint32_t rxWater)

Sets the transmit and receive FIFO watermark values.

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

- base – LPSPI peripheral address.
- txWater – The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.
- rxWater – The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated.

static inline void LPSPI_SetAllPcsPolarity(LPSPI_Type *base, uint32_t mask)

Configures all LPSPI peripheral chip select polarities simultaneously.

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx_CR_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow | kLPSPI_Pcs1ActiveLow);
```

Parameters

- base – LPSPI peripheral address.
- mask – The PCS polarity mask; Use the enum _lpspi_pcs_polarity.

static inline void LPSPI_SetFrameSize(LPSPI_Type *base, uint32_t frameSize)

Configures the frame size.

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

- base – LPSPI peripheral address.
- frameSize – The frame size in number of bits.

uint32_t LPSPI_MasterSetBaudRate(LPSPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *tcrPrescaleValue)

Sets the LPSPI baud rate in bits per second.

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate

in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale tcrPrescaleValue parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- baudRate_Bps – The desired baud rate in bits per second.
- srcClock_Hz – Module source input clock in Hertz.
- tcrPrescaleValue – The TCR prescale value needed to program the TCR.

Returns

The actual calculated baud rate. This function may also return a "0" if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

void LPSPI_MasterSetDelayScaler(LPSPI_Type *base, uint32_t scaler, *lpspi_delay_type_t* whichDelay)

Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type lpspi_delay_type_t.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.
- scaler – The 8-bit delay value 0x00 to 0xFF (255).
- whichDelay – The desired delay to configure, must be of type lpspi_delay_type_t.

uint32_t LPSPI_MasterSetDelayTimes(LPSPI_Type *base, uint32_t delayTimeInNanoSec, *lpspi_delay_type_t* whichDelay, uint32_t srcClock_Hz)

Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).

This function configures the SCK to PCS delay, PCS to SCK delay, or the delay between transfers. The delay names are available in type lpspi_delay_type_t.

The function calculates the value needed for the desired delay parameter and returns the actual calculated delay. An exact delay match may not be possible, in which case the closest match is calculated without going below the desired delay value. If the input exceeds the maximum capability, the maximum supported delay is returned.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

- base – LPSPI peripheral address.

- delayTimeInNanoSec – The desired delay value in nanoseconds.
- whichDelay – The desired delay to configure, must be of type lpspi_delay_type_t.
- srcClock_Hz – Module source input clock in Hertz.

Returns
Actual calculated delay value in nanoseconds.

static inline void LPSPI_WriteData(LPSPI_Type *base, uint32_t data)

Writes data into the transmit data buffer.

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

- base – LPSPI peripheral address.
- data – The data word to be sent.

static inline uint32_t LPSPI_ReadData(LPSPI_Type *base)

Reads data from the data buffer.

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

- base – LPSPI peripheral address.

Returns
The data read from the data buffer.

void LPSPI_SetDummyData(LPSPI_Type *base, uint8_t dummyData)

Set up the dummy data.

Parameters

- base – LPSPI peripheral address.
- dummyData – Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated.

void LPSPI_MasterTransferCreateHandle(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_master_transfer_callback_t* callback, void *userData)

Initializes the LPSPI master handle.

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_master_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_MasterTransferBlocking(LPSPI_Type *base, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using a polling method.

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferNonBlocking(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using an interrupt method.

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferGetCount(LPSPI_Type *base, *lpspi_master_handle_t* *handle, size_t *count)

Gets the master transfer remaining bytes.

This function gets the master transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Returns

status of status_t.

void LPSPI_MasterTransferAbort(LPSPI_Type *base, *lpspi_master_handle_t* *handle)

LPSPI master abort transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_MasterTransferHandleIRQ(uint32_t instance, *lpspi_master_handle_t* *handle)

LPSPI Master IRQ handler function.

This function processes the LPSPI transmit and receive IRQ.

Parameters

- instance – LPSPI instance.
- handle – pointer to lpspi_master_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferCreateHandle(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_slave_transfer_callback_t* callback, void *userData)

Initializes the LPSPI slave handle.

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

- base – LPSPI peripheral address.
- handle – LPSPI handle pointer to lpspi_slave_handle_t.
- callback – DSPI callback.
- userData – callback function parameter.

status_t LPSPI_SlaveTransferNonBlocking(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfer data using an interrupt method.

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_SlaveTransferGetCount(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes.

This function gets the slave transfer remaining bytes.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

- count – Number of bytes transferred so far by the non-blocking transaction.

Returns
status of status_t.

void LPSPI_SlaveTransferAbort(LPSPI_Type *base, *lpspi_slave_handle_t* *handle)

LPSPI slave aborts a transfer which uses an interrupt method.

This function aborts a transfer which uses an interrupt method.

Parameters

- base – LPSPI peripheral address.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

void LPSPI_SlaveTransferHandleIRQ(uint32_t instance, *lpspi_slave_handle_t* *handle)

LPSPI Slave IRQ handler function.

This function processes the LPSPI transmit and receives an IRQ.

Parameters

- instance – LPSPI instance index.
- handle – pointer to lpspi_slave_handle_t structure which stores the transfer state.

FSL_LPSPI_DRIVER_VERSION

LPSPI driver version.

Status for the LPSPI driver.

Values:

enumerator kStatus_LPSPI_Busy

LPSPI transfer is busy.

enumerator kStatus_LPSPI_Error

LPSPI driver error.

enumerator kStatus_LPSPI_Idle

LPSPI is idle.

enumerator kStatus_LPSPI_OutOfRange

LPSPI transfer out Of range.

enumerator kStatus_LPSPI_Timeout

LPSPI timeout polling status flags.

enum _lpspi_flags

LPSPI status flags in SPIx_SR register.

Values:

enumerator kLPSPI_TxDataRequestFlag

Transmit data flag

enumerator kLPSPI_RxDataReadyFlag
Receive data flag

enumerator kLPSPI_WordCompleteFlag
Word Complete flag

enumerator kLPSPI_FrameCompleteFlag
Frame Complete flag

enumerator kLPSPI_TransferCompleteFlag
Transfer Complete flag

enumerator kLPSPI_TransmitErrorFlag
Transmit Error flag (FIFO underrun)

enumerator kLPSPI_ReceiveErrorFlag
Receive Error flag (FIFO overrun)

enumerator kLPSPI_DataMatchFlag
Data Match flag

enumerator kLPSPI_ModuleBusyFlag
Module Busy flag

enumerator kLPSPI_AllStatusFlag
Used for clearing all w1c status flags

enum _lpspi_interrupt_enable
LPSPI interrupt source.
Values:

enumerator kLPSPI_TxInterruptEnable
Transmit data interrupt enable

enumerator kLPSPI_RxInterruptEnable
Receive data interrupt enable

enumerator kLPSPI_WordCompleteInterruptEnable
Word complete interrupt enable

enumerator kLPSPI_FrameCompleteInterruptEnable
Frame complete interrupt enable

enumerator kLPSPI_TransferCompleteInterruptEnable
Transfer complete interrupt enable

enumerator kLPSPI_TransmitErrorInterruptEnable
Transmit error interrupt enable(FIFO underrun)

enumerator kLPSPI_ReceiveErrorInterruptEnable
Receive Error interrupt enable (FIFO overrun)

enumerator kLPSPI_DataMatchInterruptEnable
Data Match interrupt enable

enumerator kLPSPI_AllInterruptEnable
All above interrupts enable.

enum _lpspi_dma_enable
LPSPI DMA source.
Values:

enumerator kLPSPI_TxDmaEnable
Transmit data DMA enable

enumerator kLPSPI_RxDmaEnable
Receive data DMA enable

enum _lpspi_master_slave_mode
LPSPI master or slave mode configuration.
Values:

enumerator kLPSPI_Master
LPSPI peripheral operates in master mode.

enumerator kLPSPI_Slave
LPSPI peripheral operates in slave mode.

enum _lpspi_which_pcs_config
LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).
Values:

enumerator kLPSPI_Pcs0
PCS[0]

enumerator kLPSPI_Pcs1
PCS[1]

enumerator kLPSPI_Pcs2
PCS[2]

enumerator kLPSPI_Pcs3
PCS[3]

enum _lpspi_pcs_polarity_config
LPSPI Peripheral Chip Select (PCS) Polarity configuration.
Values:

enumerator kLPSPI_PcsActiveHigh
PCS Active High (idles low)

enumerator kLPSPI_PcsActiveLow
PCS Active Low (idles high)

enum _lpspi_pcs_polarity
LPSPI Peripheral Chip Select (PCS) Polarity.
Values:

enumerator kLPSPI_Pcs0ActiveLow
Pcs0 Active Low (idles high).

enumerator kLPSPI_Pcs1ActiveLow
Pcs1 Active Low (idles high).

enumerator kLPSPI_Pcs2ActiveLow
Pcs2 Active Low (idles high).

enumerator kLPSPI_Pcs3ActiveLow
Pcs3 Active Low (idles high).

enumerator kLPSPI_PcsAllActiveLow
Pcs0 to Pcs5 Active Low (idles high).

enum _lpspi_clock_polarity
LPSPI clock polarity configuration.
Values:

enumerator kLPSPI_ClockPolarityActiveHigh
CPOL=0. Active-high LPSPI clock (idles low)

enumerator kLPSPI_ClockPolarityActiveLow
CPOL=1. Active-low LPSPI clock (idles high)

enum _lpspi_clock_phase
LPSPI clock phase configuration.
Values:

enumerator kLPSPI_ClockPhaseFirstEdge
CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kLPSPI_ClockPhaseSecondEdge
CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum _lpspi_shift_direction

LPSPI data shifter direction options.

Values:

enumerator kLPSPI_MsbFirst

Data transfers start with most significant bit.

enumerator kLPSPI_LsbFirst

Data transfers start with least significant bit.

enum _lpspi_host_request_select

LPSPI Host Request select configuration.

Values:

enumerator kLPSPI_HostReqExtPin

Host Request is an ext pin.

enumerator kLPSPI_HostReqInternalTrigger

Host Request is an internal trigger.

enum _lpspi_match_config

LPSPI Match configuration options.

Values:

enumerator kLPSI_MatchDisabled

LPSPI Match Disabled.

enumerator kLPSI_1stWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0orM1

LPSPI Match Enabled.

enumerator kLPSI_1stWordEqualsM0and2ndWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSI_AnyWordEqualsM0andNxtWordEqualsM1

LPSPI Match Enabled.

enumerator kLPSI_1stWordAndM1EqualsM0andM1

LPSPI Match Enabled.

enumerator kLPSI_AnyWordAndM1EqualsM0andM1

LPSPI Match Enabled.

enum _lpspi_pin_config

LPSPI pin (SDO and SDI) configuration.

Values:

enumerator kLPSPI_SdiInSdoOut
LPSPI SDI input, SDO output.

enumerator kLPSPI_SdiInSdiOut
LPSPI SDI input, SDI output.

enumerator kLPSPI_SdoInSdoOut
LPSPI SDO input, SDO output.

enumerator kLPSPI_SdoInSdiOut
LPSPI SDO input, SDI output.

enum _lpspi_data_out_config
LPSPI data output configuration.
Values:

enumerator kLpspiDataOutRetained
Data out retains last value when chip select is de-asserted

enumerator kLpspiDataOutTristate
Data out is tristated when chip select is de-asserted

enum _lpspi_pcs_function_config
LPSPI cs function configuration.
Values:

enumerator kLPSPI_PcsAsCs
PCS pin select as cs function

enumerator kLPSPI_PcsAsData
PCS pin select as date function

enum _lpspi_transfer_width
LPSPI transfer width configuration.
Values:

enumerator kLPSPI_SingleBitXfer
1-bit shift at a time, data out on SDO, in on SDI (normal mode)

enumerator kLPSPI_TwoBitXfer
2-bits shift out on SDO/SDI and in on SDO/SDI

enumerator kLPSPI_FourBitXfer
4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

enum _lpspi_delay_type
LPSPI delay type selection.
Values:

enumerator kLPSPI_PcsToSck

PCS-to-SCK delay.

enumerator kLPSPI_LastSckToPcs

Last SCK edge to PCS delay.

enumerator kLPSPI_BetweenTransfer

Delay between transfers.

enum _lpspi_transfer_config_flag_for_master

Use this enumeration for LPSPI master transfer configFlags.

Values:

enumerator kLPSPI_MasterPcs0

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS0 signal

enumerator kLPSPI_MasterPcs1

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS1 signal

enumerator kLPSPI_MasterPcs2

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS2 signal

enumerator kLPSPI_MasterPcs3

LPSPI master PCS shift macro , internal used. LPSPI master transfer use PCS3 signal

enumerator kLPSPI_MasterWidth1

LPSPI master width shift macro, internal used LPSPI master transfer 1bit

enumerator kLPSPI_MasterWidth2

LPSPI master width shift macro, internal used LPSPI master transfer 2bit

enumerator kLPSPI_MasterWidth4

LPSPI master width shift macro, internal used LPSPI master transfer 4bit

enumerator kLPSPI_MasterPcsContinuous

Is PCS signal continuous

enumerator kLPSPI_MasterByteSwap

Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_MasterByteSwapyou flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_MasterByteSwap flag.

enum _lpspi_transfer_config_flag_for_slave

Use this enumeration for LPSPI slave transfer configFlags.

Values:

enumerator kLPSPI_SlavePcs0

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS0 signal

enumerator kLPSPI_SlavePcs1

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS1 signal

enumerator kLPSPI_SlavePcs2

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS2 signal

enumerator kLPSPI_SlavePcs3

LPSPI slave PCS shift macro , internal used. LPSPI slave transfer use PCS3 signal

enumerator kLPSPI_SlaveByteSwap

Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi_shift_direction_t to MSB).

i. If you set bitPerFrame = 8 , no matter the kLPSPI_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.

ii. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

iii. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI_SlaveByteSwap flag.

enum _lpspi_transfer_state

LPSPI transfer state, which is used for LPSPI transactional API state machine.

Values:

enumerator kLPSPI_Idle

Nothing in the transmitter/receiver.

enumerator kLPSPI_Busy

Transfer queue is not finished.

enumerator kLPSPI_Error

Transfer error.

typedef enum *_lpspi_master_slave_mode* lpspi_master_slave_mode_t

LPSPI master or slave mode configuration.

typedef enum *_lpspi_which_pcs_config* lpspi_which_pcs_t

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).

typedef enum *_lpspi_pcs_polarity_config* lpspi_pcs_polarity_config_t
LPSPI Peripheral Chip Select (PCS) Polarity configuration.

typedef enum *_lpspi_clock_polarity* lpspi_clock_polarity_t
LPSPI clock polarity configuration.

typedef enum *_lpspi_clock_phase* lpspi_clock_phase_t
LPSPI clock phase configuration.

typedef enum *_lpspi_shift_direction* lpspi_shift_direction_t
LPSPI data shifter direction options.

typedef enum *_lpspi_host_request_select* lpspi_host_request_select_t
LPSPI Host Request select configuration.

typedef enum *_lpspi_match_config* lpspi_match_config_t
LPSPI Match configuration options.

typedef enum *_lpspi_pin_config* lpspi_pin_config_t
LPSPI pin (SDO and SDI) configuration.

typedef enum *_lpspi_data_out_config* lpspi_data_out_config_t
LPSPI data output configuration.

typedef enum *_lpspi_pcs_function_config* lpspi_pcs_function_config_t
LPSPI cs function configuration.

typedef enum *_lpspi_transfer_width* lpspi_transfer_width_t
LPSPI transfer width configuration.

typedef enum *_lpspi_delay_type* lpspi_delay_type_t
LPSPI delay type selection.

typedef struct *_lpspi_master_config* lpspi_master_config_t
LPSPI master configuration structure.

typedef struct *_lpspi_slave_config* lpspi_slave_config_t
LPSPI slave configuration structure.

typedef struct *_lpspi_master_handle* lpspi_master_handle_t
Forward declaration of the *_lpspi_master_handle* typedefs.

typedef struct *_lpspi_slave_handle* lpspi_slave_handle_t
Forward declaration of the *_lpspi_slave_handle* typedefs.

typedef void (*lpspi_master_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_handle_t* *handle, *status_t* status, void *userData)
Master completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI master.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_handle_t* *handle, *status_t* status, void *userData)

Slave completion callback function pointer type.

Param base
LPSPI peripheral address.

Param handle
Pointer to the handle for the LPSPI slave.

Param status
Success or error code describing whether the transfer is completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef struct *_lpspi_transfer* lpspi_transfer_t

LPSPI master/slave transfer structure.

volatile uint8_t g_lpspiDummyData[]

Global variable for dummy data value setting.

LPSPI_DUMMY_DATA

LPSPI dummy data if no Tx data.

Dummy data used for tx if there is not txData.

SPI_RETRY_TIMES

Retry times for waiting flag.

LPSPI_MASTER_PCS_SHIFT

LPSPI master PCS shift macro , internal used.

LPSPI_MASTER_PCS_MASK

LPSPI master PCS shift macro , internal used.

LPSPI_MASTER_WIDTH_SHIFT

LPSPI master width shift macro, internal used

LPSPI_MASTER_WIDTH_MASK

LPSPI master width shift mask, internal used

LPSPI_SLAVE_PCS_SHIFT

LPSPI slave PCS shift macro , internal used.

LPSPI_SLAVE_PCS_MASK

LPSPI slave PCS shift macro , internal used.

struct _lpspi_master_config

#include <fsl_lpspi.h> LPSPI master configuration structure.

Public Members

uint32_t baudRate

Baud Rate for LPSPI.

uint32_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol

Clock polarity.

lpspi_clock_phase_t cpha

Clock phase.

lpspi_shift_direction_t direction

MSB or LSB data shift direction.

uint32_t pcsToSckDelayInNanoSec

PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t lastSckToPcsDelayInNanoSec

Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

uint32_t betweenTransferDelayInNanoSec

After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay. It sets the boundary value if out of range.

lpspi_which_pcs_t whichPcs

Desired Peripheral Chip Select (PCS).

lpspi_pcs_polarity_config_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi_pin_config_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi_pcs_function_config_t pcsFunc

Configures cs pins function.

lpspi_data_out_config_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

bool enableInputDelay

Enable master to sample the input data on a delayed SCK. This can help improve slave setup time. Refer to device data sheet for specific time length.

struct __lpspi__slave__config

#include <fsl_lpspi.h> LPSPI slave configuration structure.

Public Members

uint32_t bitsPerFrame

Bits per frame, minimum 8, maximum 4096.

lpspi_clock_polarity_t cpol

Clock polarity.

lpspi_clock_phase_t cpha

Clock phase.

lpspi_shift_direction_t direction

MSB or LSB data shift direction.

lpspi_which_pcs_t whichPcs

Desired Peripheral Chip Select (pcs)

lpspi_pcs_polarity_config_t pcsActiveHighOrLow

Desired PCS active high or low

lpspi_pin_config_t pinCfg

Configures which pins are used for input and output data during single bit transfers.

lpspi_data_out_config_t dataOutConfig

Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

struct __lpspi__transfer

#include <fsl_lpspi.h> LPSPI master/slave transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

volatile size_t dataSize
: Transfer bytes.

uint32_t configFlags
: Transfer transfer configuration flags. Set from _lpspi_transfer_config_flag_for_master if the transfer is used for master or _lpspi_transfer_config_flag_for_slave enumeration if the transfer is used for slave.

struct __lpspi_master_handle
: *#include <fsl_lpspi.h>* LPSPI master transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous
: Is PCS continuous in transfer.

volatile bool writeTcrInIsr
: A flag that whether should write TCR in ISR.

volatile bool isByteSwap
: A flag that whether should byte swap.

volatile bool isTxMask
: A flag that whether TCR[TXMSK] is set.

volatile uint16_t bytesPerFrame
: Number of bytes in each frame

volatile uint8_t fifoSize
: FIFO dataSize.

volatile uint8_t rxWatermark
: Rx watermark.

volatile uint8_t bytesEachWrite
: Bytes for each write TDR.

volatile uint8_t bytesEachRead
: Bytes for each read RDR.

const uint8_t *volatile txData
: Send buffer.

uint8_t *volatile rxData
: Receive buffer.

volatile size_t txRemainingByteCount
: Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
: Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
: Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
: Read RDR register remaining times.

uint32_t totalByteCount
: Number of transfer bytes

uint32_t txBuffIfNull
: Used if the txData is NULL.

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

lpspi_master_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

struct __lpspi_slave_handle
: *#include <fsl_lpspi.h>* LPSPI slave transfer handle structure used for transactional API.

Public Members

volatile bool isByteSwap
: A flag that whether should byte swap.

volatile uint8_t fifoSize
: FIFO dataSize.

volatile uint8_t rxWatermark
: Rx watermark.

volatile uint8_t bytesEachWrite
: Bytes for each write TDR.

volatile uint8_t bytesEachRead
: Bytes for each read RDR.

const uint8_t *volatile txData
: Send buffer.

uint8_t *volatile rxData
Receive buffer.

volatile size_t txRemainingByteCount
Number of bytes remaining to send.

volatile size_t rxRemainingByteCount
Number of bytes remaining to receive.

volatile uint32_t writeRegRemainingTimes
Write TDR register remaining times.

volatile uint32_t readRegRemainingTimes
Read RDR register remaining times.

uint32_t totalByteCount
Number of transfer bytes

volatile uint8_t state
LPSPI transfer state , _lpspi_transfer_state.

volatile uint32_t errorCount
Error count for slave transfer.

lpspi_slave_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

2.56 LPSPI eDMA Driver

FSL_LPSPI_EDMA_DRIVER_VERSION
LPSPI EDMA driver version.

typedef struct *_lpspi_master_edma_handle* lpspi_master_edma_handle_t
Forward declaration of the *_lpspi_master_edma_handle* typedefs.

typedef struct *_lpspi_slave_edma_handle* lpspi_slave_edma_handle_t
Forward declaration of the *_lpspi_slave_edma_handle* typedefs.

typedef void (*lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *status_t* status, void *userData)
Completion callback function pointer type.

Param base
LPSPI peripheral base address.

Param handle
Pointer to the handle for the LPSPI master.

Param status
Success or error code describing whether the transfer completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef void (*lpspi_slave_edma_transfer_callback_t)(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base
LPSPI peripheral base address.

Param handle
Pointer to the handle for the LPSPI slave.

Param status
Success or error code describing whether the transfer completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

void LPSPI_MasterTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI master eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that the LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Tx DMAMUX source for edmaRxRegToRxDataHandle.

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to lpspi_master_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t LPSPI_MasterTransferEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size

should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

status_t LPSPI_MasterTransferPrepareEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, uint32_t configFlags)

LPSPI master config transfer parameter while using eDMA.

This function is preparing to transfer data using eDMA, work with LPSPI_MasterTransferEDMALite.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- configFlags – transfer configuration flags. *_lpspi_transfer_config_flag_for_master*.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.

Returns

Indicates whether LPSPI master transfer was successful or not.

status_t LPSPI_MasterTransferEDMALite(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI master transfer data using eDMA without configs.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: This API is only for transfer through DMA without configuration. Before calling this API, you must call LPSPI_MasterTransferPrepareEDMALite to configure it once. The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure, config field is not uesed.

Return values

- kStatus_Success – Execution successfully.
- kStatus_LPSPI_Busy – The LPSPI device is busy.

- kStatus_InvalidArgument – The transfer structure is invalid.

Returns
Indicates whether LPSPI master transfer was successful or not.

void LPSPI_MasterTransferAbortEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle)

LPSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.

status_t LPSPI_MasterTransferGetCountEDMA(LPSPI_Type *base, *lpspi_master_edma_handle_t* *handle, size_t *count)

Gets the master eDMA transfer remaining bytes.

This function gets the master eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_master_edma_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the EDMA transaction.

Returns
status of status_t.

void LPSPI_SlaveTransferCreateHandleEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_slave_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the LPSPI slave eDMA handle.

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle .

Parameters

- base – LPSPI peripheral base address.
- handle – LPSPI handle pointer to lpspi_slave_edma_handle_t.
- callback – LPSPI callback.
- userData – callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.

- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t LPSPI_SlaveTransferEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, *lpspi_transfer_t* *transfer)

LPSPI slave transfers data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- transfer – pointer to lpspi_transfer_t structure.

Returns

status of status_t.

void LPSPI_SlaveTransferAbortEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle)

LPSPI slave aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.

status_t LPSPI_SlaveTransferGetCountEDMA(LPSPI_Type *base, *lpspi_slave_edma_handle_t* *handle, size_t *count)

Gets the slave eDMA transfer remaining bytes.

This function gets the slave eDMA transfer remaining bytes.

Parameters

- base – LPSPI peripheral base address.
- handle – pointer to lpspi_slave_edma_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the eDMA transaction.

Returns

status of status_t.

struct __lpspi_master_edma_handle

#include <fsl_lpspi_edma.h> LPSPI master eDMA transfer handle structure used for transactional API.

Public Members

volatile bool isPcsContinuous
: Is PCS continuous in transfer.

volatile bool isByteSwap
: A flag that whether should byte swap.

uint32_t totalByteCount
: Number of transfer bytes

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

uint32_t txBuffIfNull
: Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
: Used if there is not rxData for DMA purpose.

uint32_t transmitCommand
: Used to write TCR for DMA purpose.

volatile int8_t oneFifoBlockRxWatermark
: Used to change RXWATER in FCR for DMA purpose.

volatile int8_t lastBytesRxWatermark
: Used to change RXWATER in FCR for DMA purpose.

volatile uint8_t state
: LPSPI transfer state , _lpspi_transfer_state.

lpspi_master_edma_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
: edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
: edma_handle_t handle point used for TxData to TxReg buff

edma_tcd_t lpspiSoftwareTCD[6]
: SoftwareTCD, internal used

struct __lpspi_slave_edma_handle
: *#include <fsl_lpspi_edma.h>* LPSPI slave eDMA transfer handle structure used for transactional API.

Public Members

uint32_t totalByteCount
: Number of transfer bytes

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

uint32_t txBuffIfNull
: Used if there is not txData for DMA purpose.

uint32_t rxBuffIfNull
: Used if there is not rxData for DMA purpose.

volatile int8_t oneFifoBlockRxWatermark
: Used to change RXWATER in FCR for DMA purpose.

volatile int8_t lastBytesRxWatermark
: Used to change RXWATER in FCR for DMA purpose.

volatile uint8_t state
: LPSPI transfer state.

lpspi_slave_edma_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
: edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
: edma_handle_t handle point used for TxData to TxReg

edma_tcd_t lpspiSoftwareTCD[5]
: SoftwareTCD, internal used

2.57 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.58 LPUART Driver

static inline void LPUART_SoftwareReset(LPUART_Type *base)
: Resets the LPUART using software.

  This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

- base – LPUART peripheral base address.

status_t LPUART_Init(LPUART_Type *base, const *lpuart_config_t* *config, uint32_t srcClock_Hz)

Initializes an LPUART instance with the user configuration structure and the peripheral clock.

This function configures the LPUART module with user-defined settings. Call the *LPUART_GetDefaultConfig()* function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
lpuart_config_t lpuartConfig;
lpuartConfig.baudRate_Bps = 115200U;
lpuartConfig.parityMode = kLPUART_ParityDisabled;
lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
lpuartConfig.isMsb = false;
lpuartConfig.stopBitCount = kLPUART_OneStopBit;
lpuartConfig.txFifoWatermark = 0;
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- config – Pointer to a user-defined configuration structure.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – LPUART initialize succeed

void LPUART_Deinit(LPUART_Type *base)

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

- base – LPUART peripheral base address.

void LPUART_GetDefaultConfig(*lpuart_config_t* *config)

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

- config – Pointer to a configuration structure.

status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- baudRate_Bps – LPUART baudrate to be set.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not supported in the current clock source.
- kStatus_Success – Set baudrate succeeded.

void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- base – LPUART peripheral base address.
- enable – true to enable, flase to disable.

static inline void LPUART_SetMatchAddress(LPUART_Type *base, uint16_t address1, uint16_t address2)

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

static inline void LPUART_EnableMatchAddress(LPUART_Type *base, bool match1, bool match2)

Enable the LPUART match address feature.

Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void LPUART_SetRxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

static inline void LPUART_SetTxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

static inline void LPUART_TransferEnable16Bit(*lpuart_handle_t* *handle, bool enable)

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in lpuart_handle_t.

Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

uint32_t LPUART_GetStatusFlags(LPUART_Type *base)

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators *_lpuart_flags*. To check for a specific status, compare the return value with enumerators in the *_lpuart_flags*. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART_GetStatusFlags(LPUART1))
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the _lpuart_flags.

status_t LPUART_ClearStatusFlags(LPUART_Type *base, uint32_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART_TxDataRegEmptyFlag, kLPUART_TransmissionCompleteFlag, kLPUART_RxDataRegFullFlag, kLPUART_RxActiveFlag, kLPUART_NoiseErrorInRxDataRegFlag, kLPUART_ParityErrorInRxDataRegFlag, kLPUART_TxFifoEmptyFlag,kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- base – LPUART peripheral base address.

- mask – the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask.

Return values

- kStatus_LPUART_FlagCannotClearManually – The flag can't be cleared by this function but it is cleared automatically by hardware.
- kStatus_Success – Status in the mask are cleared.

Returns

0 succeed, others failed.

void LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the *_lpuart_interrupt_enable*. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of the enumeration _uart_interrupt_enable.

void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See *_lpuart_interrupt_enable*. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of *_lpuart_interrupt_enable*.

uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators *_lpuart_interrupt_enable*. To check a specific interrupt enable status, compare the return value with enumerators in *_lpuart_interrupt_enable*. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns
LPUART interrupt flags which are logical OR of the enumerators in *_lpuart_interrupt_enable*.

static inline uint32_t LPUART_GetDataRegisterAddress(LPUART_Type *base)

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters
- base – LPUART peripheral base address.

Returns
LPUART data register addresses which are used both by the transmitter and receiver.

static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters
- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters
- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

uint32_t LPUART_GetInstance(LPUART_Type *base)

Get the LPUART instance from peripheral base address.

Parameters
- base – LPUART peripheral base address.

Returns
LPUART instance.

static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters
- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t LPUART_ReadByte(LPUART_Type *base)

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns

Data read from data register.

static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns

tx FIFO data count.

void LPUART_SendAddress(LPUART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

Parameters

- base – LPUART peripheral base address.
- address – LPUART slave address.

status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.

- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_WriteBlocking16bit(LPUART_Type *base, const uint16_t *data, size_t length)

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_ReadBlocking(LPUART_Type *base, uint8_t *data, size_t length)

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t LPUART_ReadBlocking16bit(LPUART_Type *base, uint16_t *data, size_t length)

Reads the receiver data register in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

void LPUART_TransferCreateHandle(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_callback_t* callback, void *userData)

Initializes the LPUART handle.

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the *LPUART_TransferReceiveNonBlocking()* API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- callback – Callback function.
- userData – User data.

status_t LPUART_TransferSendNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the *kStatus_LPUART_TxIdle* as status parameter.

Note: The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

- base – LPUART peripheral base address.

- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see *lpuart_transfer_t*.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_LPUART_TxBusy – Previous transmission still not finished, data not all written to the TX register.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferStartRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, then only 31 bytes are used for saving data.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

void LPUART_TransferStopRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

size_t LPUART_TransferGetRxRingBufferLength(LPUART_Type *base, *lpuart_handle_t* *handle)

Get the length of received data in RX ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

void LPUART_TransferAbortSend(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetSendCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferReceiveNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to xfer->data, which returns with the parameter receivedBytes set to 5. For the remaining 5 bytes, the newly arrived data is saved from xfer->data[5]. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_LPUART_RxBusy – Previous receive request is not finished.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortReceive(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetReceiveCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferHandleIRQ(uint32_t instance, void *irqHandle)

LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- instance – LPUART instance.
- irqHandle – LPUART handle pointer.

void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)

LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

FSL_LPUART_DRIVER_VERSION

LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator kStatus_LPUART_TxBusy

TX busy

enumerator kStatus_LPUART_RxBusy

RX busy

enumerator kStatus_LPUART_TxIdle

LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle
LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge
TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge
RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually
Some flag can't manually clear

enumerator kStatus_LPUART_Error
Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun
LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun
LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError
LPUART noise error.

enumerator kStatus_LPUART_FramingError
LPUART framing error.

enumerator kStatus_LPUART_ParityError
LPUART parity error.

enumerator kStatus_LPUART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_LPUART_IdleLineDetected
IDLE flag.

enumerator kStatus_LPUART_Timeout
LPUART times out.

enum _lpuart_parity_mode
LPUART parity mode.
Values:

enumerator kLPUART_ParityDisabled
Parity disabled

enumerator kLPUART_ParityEven
Parity enabled, type even, bit setting: PE|PT = 10

enumerator kLPUART_ParityOdd
Parity enabled, type odd, bit setting: PE|PT = 11

enum _lpuart_data_bits
LPUART data bits count.
Values:

enumerator kLPUART_EightDataBits
Eight data bit

enumerator kLPUART_SevenDataBits
Seven data bit

enum _lpuart_stop_bit_count
LPUART stop bit count.
Values:

enumerator kLPUART_OneStopBit
One stop bit

enumerator kLPUART_TwoStopBit
Two stop bits

enum _lpuart_transmit_cts_source
LPUART transmit CTS source.
Values:

enumerator kLPUART_CtsSourcePin
CTS resource is the LPUART_CTS pin.

enumerator kLPUART_CtsSourceMatchResult
CTS resource is the match result.

enum _lpuart_transmit_cts_config
LPUART transmit CTS configure.
Values:

enumerator kLPUART_CtsSampleAtStart
CTS input is sampled at the start of each character.

enumerator kLPUART_CtsSampleAtIdle
CTS input is sampled when the transmitter is idle

enum _lpuart_idle_type_select
LPUART idle flag type defines when the receiver starts counting.
Values:

enumerator kLPUART_IdleTypeStartBit
Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit
Start counting after a stop bit.

enum _lpuart_idle_config
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1
the number of idle characters.

enumerator kLPUART_IdleCharacter2
the number of idle characters.

enumerator kLPUART_IdleCharacter4
the number of idle characters.

enumerator kLPUART_IdleCharacter8
the number of idle characters.

enumerator kLPUART_IdleCharacter16
the number of idle characters.

enumerator kLPUART_IdleCharacter32
the number of idle characters.

enumerator kLPUART_IdleCharacter64
the number of idle characters.

enumerator kLPUART_IdleCharacter128
the number of idle characters.

enum _lpuart_interrupt_enable
LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_CtsStateChangeInterruptEnable
Change of state on CTS_B pin. bit 0

enumerator kLPUART_DsrStateChangeInterruptEnable
Change of state on DSR_B pin. bit 1

enumerator kLPUART_RinStateChangeInterruptEnable
Change of state on RIN_B pin. bit 2

enumerator kLPUART_DcdStateChangeInterruptEnable
Change of state on DCD_B pin. bit 3

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge. bit 6

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect. bit 7

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxCounter0TimeoutInterruptEnable
Receiver counter0 timeout. bit 10

enumerator kLPUART_RxCounter1TimeoutInterruptEnable
Receiver counter1 timeout. bit 11

enumerator kLPUART_TxCounter0TimeoutInterruptEnable
Transmitter counter0 timeout. bit 12

enumerator kLPUART_TxCounter1TimeoutInterruptEnable
Transmitter counter1 timeout. bit 13

enumerator kLPUART_DataMatch2InterruptEnable
The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_DataMatch1InterruptEnable
The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_IdleLineInterruptEnable
Idle line. bit 20

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full. bit 21

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete. bit 22

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty. bit 23

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag. bit 24

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag. bit 25

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag. bit 26

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun. bit 27

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags
LPUART status flags.
This provides constants for the LPUART status flags for use in the LPUART functions.
Values:

enumerator kLPUART_RxFifoUnderflowFlag
RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART_TxFifoOverflowFlag
TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART_RxFifoEmptyFlag
RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART_TxFifoEmptyFlag
TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART_CtsStateChangeFlag
Change of state on CTS_B pin. bit 2

enumerator kLPUART_DsrStateChangeFlag
Change of state on DSR_B pin. bit 3

enumerator kLPUART_RinStateChangeFlag
Change of state on RIN_B pin. bit 4

enumerator kLPUART_DcdStateChangeFlag
Change of state on DCD_B pin. bit 5

enumerator kLPUART_RxCounter0TimeoutFlag
Receiver counter0 timeout. bit 10

enumerator kLPUART_RxCounter1TimeoutFlag
Receiver counter1 timeout. bit 11

enumerator kLPUART_TxCounter0TimeoutFlag
Transmitter counter0 timeout. bit 12

enumerator kLPUART_TxCounter1TimeoutFlag

Transmitter counter1 timeout. bit 13

enumerator kLPUART_DataMatch2Flag

The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_DataMatch1Flag

The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_ParityErrorFlag

If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART_FramingErrorFlag

Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_NoiseErrorFlag

Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_RxOverrunFlag

Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_IdleLineFlag

Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxDataRegFullFlag

Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_TransmissionCompleteFlag

Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_TxDataRegEmptyFlag

Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_RxActiveFlag

Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART_RxActiveEdgeFlag

Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART_LinBreakFlag

LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART_AllClearFlags

enumerator kLPUART_AllFlags

enum _lpuart_timeout_condition

LPUART timeout condition. This structure defines the conditions when the counter timeout occur.

Values:

enumerator kLPUART_TimeoutAfterCharacters

Timeout occurs when the number of characters specified by timeoutValue are received.

enumerator kLPUART_TimeoutAfterIdle

Timeout occurs when rx/tx remains idle for timeoutValue of bit clocks after idle condition is detected.

enumerator kLPUART_TimeoutAfterNext

Timeout occurs when rx/tx remains idle for timeoutValue of bit clocks after next character is received/transmitted.

enumerator kLPUART_TimeoutAfterIdleBeforeExtended

Timeout occurs when tx/rx is idle for larger than timeoutValue of bit clocks and smaller than tx/rx extended timeout value.

typedef enum *_lpuart_parity_mode* lpuart_parity_mode_t

LPUART parity mode.

typedef enum *_lpuart_data_bits* lpuart_data_bits_t

LPUART data bits count.

typedef enum *_lpuart_stop_bit_count* lpuart_stop_bit_count_t

LPUART stop bit count.

typedef enum *_lpuart_transmit_cts_source* lpuart_transmit_cts_source_t

LPUART transmit CTS source.

typedef enum *_lpuart_transmit_cts_config* lpuart_transmit_cts_config_t

LPUART transmit CTS configure.

typedef enum *_lpuart_idle_type_select* lpuart_idle_type_select_t

LPUART idle flag type defines when the receiver starts counting.

typedef enum *_lpuart_idle_config* lpuart_idle_config_t

LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

typedef enum *_lpuart_timeout_condition* lpuart_timeout_condition_t

LPUART timeout condition. This structure defines the conditions when the counter timeout occur.

typedef struct *_lpuart_timeout_counter_config* lpuart_timeout_counter_config_t

LPUART timeout counter configuration structure.

typedef struct *_lpuart_timeout_config* lpuart_timeout_config_t

LPUART timeout configuration structure.

typedef struct *_lpuart_config* lpuart_config_t

LPUART configuration structure.

typedef struct *_lpuart_transfer* lpuart_transfer_t

LPUART transfer structure.

typedef struct *_lpuart_handle* lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, *lpuart_handle_t* *handle, *status_t* status, void *userData)

LPUART transfer callback function.

typedef void (*lpuart_irq_handler_t)(uint32_t instance, void *handle)

const IRQn_Type s_lpuartIRQ[]

UART_RETRY_TIMES

Retry times for waiting flag.

struct _lpuart_timeout_counter_config

#include <fsl_lpuart.h> LPUART timeout counter configuration structure.

Public Members

bool enableCounter

Eneble the timeout counter.

lpuart_timeout_condition_t timeoutCondition

Timeout condition.

uint16_t timeoutValue

Timeout value.

struct _lpuart_timeout_config

#include <fsl_lpuart.h> LPUART timeout configuration structure.

Public Members

uint16_t rxExtendedTimeoutValue

The number of bits since the last stop bit that is required for an idle condition to be detected. Enable this will disable rxIdleType and rxIdleConfig. Set to 0 to disable.

uint16_t txExtendedTimeoutValue

The transmitter idle time in number of bits (baud rate) whenever an idle character is queued through the transmit FIFO.

lpuart_timeout_counter_config_t rxCounter0

Rx counter 0 configuration.

lpuart_timeout_counter_config_t rxCounter1

Rx counter 1 configuration.

lpuart_timeout_counter_config_t txCounter0

Tx counter 0 configuration.

lpuart_timeout_counter_config_t txCounter1

Tx counter 1 configuration.

struct __lpuart_config

#include <fsl_lpuart.h> LPUART configuration structure.

Public Members

uint32_t baudRate_Bps

LPUART baud rate

lpuart_parity_mode_t parityMode

Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount

Data bits count, eight (default), seven

bool isMsb

Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount

Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark

TX FIFO watermark

uint8_t rxFifoWatermark

RX FIFO watermark

bool enableRxRTS

RX RTS enable

bool enableTxCTS

TX CTS enable

lpuart_transmit_cts_source_t txCtsSource
: TX CTS source

lpuart_transmit_cts_config_t txCtsConfig
: TX CTS configure

lpuart_idle_type_select_t rxIdleType
: RX IDLE type.

lpuart_idle_config_t rxIdleConfig
: RX IDLE configuration.

lpuart_timeout_config_t timeoutConfig
: Timeout configuration.

bool enableSingleWire
: Use TXD pin as the source for the receiver. When enabled the TXD pin should be configured as open drain.

uint8_t rtsDelay
: Delay the negation of RTS by the configured number of bit clocks.

bool enableTx
: Enable TX

bool enableRx
: Enable RX

struct __lpuart__transfer
: *#include <fsl_lpuart.h>* LPUART transfer structure.

Public Members

size_t dataSize
: The byte count to be transfer.

struct __lpuart__handle
: *#include <fsl_lpuart.h>* LPUART handle structure.

Public Members

volatile size_t txDataSize
: Size of the remaining data to send.

size_t txDataSizeAll
: Size of the data to send out.

volatile size_t rxDataSize
Size of the remaining data to receive.

size_t rxDataSizeAll
Size of the data to receive.

size_t rxRingBufferSize
Size of the ring buffer.

volatile uint16_t rxRingBufferHead
Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
Index for the user to get data from the ring buffer.

lpuart_transfer_callback_t callback
Callback function.

void *userData
LPUART callback function parameter.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state.

bool isSevenDataBits
Seven data bits flag.

bool is16bitData
16bit data bits flag, only used for 9bit or 10bit data

union ___unnamed80___

Public Members

uint8_t *data
The buffer of data to be transfer.

uint8_t *rxData
The buffer to receive data.

uint16_t *rxData16
The buffer to receive data.

const uint8_t *txData
The buffer of data to be sent.

const uint16_t *txData16
The buffer of data to be sent.

union ___unnamed82___

Public Members

const uint8_t *volatile txData
Address of remaining data to send.

const uint16_t *volatile txData16
Address of remaining data to send.

union ___unnamed84___

Public Members

uint8_t *volatile rxData
Address of remaining data to receive.

uint16_t *volatile rxData16
Address of remaining data to receive.

union ___unnamed86___

Public Members

uint8_t *rxRingBuffer
Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16
Start address of the receiver ring buffer.

2.59 LPUART eDMA Driver

void LPUART_TransferCreateHandleEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle)

Initializes the LPUART handle which is used in transactional functions.

Note: This function disables all LPUART interrupts.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- callback – Callback function.
- userData – User data.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.

void LPUART_TransferCreateHandleEDMAExt(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle, *lpuart_tx_callback_mode_t* txCbMode)

Initializes the LPUART handle for eDMA transactional functions with extended TX callback configuration.

This function initializes the LPUART eDMA handle and allows configuration of when the TX transfer callback is triggered. Use this function when you need precise control over TX completion notification.

Note: This function disables all LPUART interrupts when TX callback mode is set to kLPUART_TxFifoEmpty.

Note: When using kLPUART_TxEdmaComplete mode, the callback occurs when eDMA finishes, but data may still be in the TX FIFO. Use kLPUART_TxFifoEmpty if you need to ensure all bytes are transmitted.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- callback – Callback function.
- userData – User data.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.
- rxEdmaHandle – User requested DMA handle for RX DMA transfer.
- txCbMode – TX callback mode. See *lpuart_tx_callback_mode_t*.

status_t LPUART_SendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART eDMA transfer structure. See *lpuart_transfer_t*.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_LPUART_TxBusy – Previous transfer on going.
- kStatus_InvalidArgument – Invalid argument.

status_t LPUART_ReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- xfer – LPUART eDMA transfer structure, see *lpuart_transfer_t*.

Return values

- kStatus_Success – if succeed, others fail.
- kStatus_LPUART_RxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortSendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

void LPUART_TransferAbortReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the received data using eDMA.

This function aborts the received data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

status_t LPUART_TransferGetSendCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferGetReceiveCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferEdmaHandleIRQ(uint32_t instance, void *lpuartEdmaHandle)

LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note: This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

- instance – LPUART peripheral index.
- lpuartEdmaHandle – LPUART handle pointer.

FSL_LPUART_EDMA_DRIVER_VERSION

LPUART EDMA driver version.

enum _lpuart_tx_callback_mode

LPUART TX transfer callback mode.

Values:

enumerator kLPUART_TxFifoEmpty

Callback is called when all bytes are transmitted (TX FIFO is empty)

enumerator kLPUART_TxEdmaComplete

Callback is called when eDMA transfer finished (data still can be in TX FIFO)

typedef enum *_lpuart_tx_callback_mode* lpuart_tx_callback_mode_t

LPUART TX transfer callback mode.

typedef struct *_lpuart_edma_handle* lpuart_edma_handle_t

typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *status_t* status, void *userData)

LPUART transfer callback function.

struct __lpuart_edma_handle

#include <fsl_lpuart_edma.h> LPUART eDMA handle.

Public Members

lpuart_edma_transfer_callback_t callback

Callback function.

void *userData

LPUART callback function parameter.

size_t rxDataSizeAll

Size of the data to receive.

size_t txDataSizeAll

Size of the data to send out.

const uint8_t *rxData

Address of RX data buffer, used for calculation of received data

const uint8_t *txData

Address of TX data buffer, used for calculation of written data

edma_handle_t *txEdmaHandle

The eDMA TX channel used.

edma_handle_t *rxEdmaHandle

The eDMA RX channel used.

volatile uint8_t txState

TX transfer state

volatile uint8_t rxState

RX transfer state

lpuart_tx_callback_mode_t txCbMode

TX transfer callback mode.

uint8_t rxWatermarkValue

RXWATER byte value written by DMA TCD when switching to remaining bytes

edma_tcd_t edmaTcd[4]

eDMA TCDs for internal use

2.60 MIPI DSI Driver

void DSI_Init(MIPI_DSI_HOST_Type *base, const *dsi_config_t* *config)

Initializes an MIPI DSI host with the user configuration.

This function initializes the MIPI DSI host with the configuration, it should be called first before other MIPI DSI driver functions.

Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to a user-defined configuration structure.

void DSI_Deinit(MIPI_DSI_HOST_Type *base)

Deinitializes an MIPI DSI host.

This function should be called after all bother MIPI DSI driver functions.

Parameters

- base – MIPI DSI host peripheral base address.

void DSI_GetDefaultConfig(*dsi_config_t* *config)

Get the default configuration to initialize the MIPI DSI host.

The default value is:

```
config->numLanes = 4;
config->enableNonContinuousHsClk = false;
config->enableTxUlps = false;
config->autoInsertEoTp = true;
config->numExtraEoTp = 0;
config->htxTo_ByteClk = 0;
config->lrxHostTo_ByteClk = 0;
config->btaTo_ByteClk = 0;
```

Parameters

- config – Pointer to a user-defined configuration structure.

void DSI_SetDpiConfig(MIPI_DSI_HOST_Type *base, const *dsi_dpi_config_t* *config, uint8_t numLanes, uint32_t dpiPixelClkFreq_Hz, uint32_t dsiHsBitClkFreq_Hz)

Configure the DPI interface core.

This function sets the DPI interface configuration, it should be used in video mode.

Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the DPI interface configuration.
- numLanes – Lane number, should be same with the setting in *dsi_dpi_config_t*.
- dpiPixelClkFreq_Hz – The DPI pixel clock frequency in Hz.
- dsiHsBitClkFreq_Hz – The DSI high speed bit clock frequency in Hz. It is the same with DPHY PLL output.

uint32_t DSI_InitDphy(MIPI_DSI_HOST_Type *base, const *dsi_dphy_config_t* *config, uint32_t refClkFreq_Hz)

Initializes the D-PHY.

This function configures the D-PHY timing and setups the D-PHY PLL based on user configuration. The configuration structure could be got by the function *DSI_GetDphyDefaultConfig*.

For some platforms there is not dedicated D-PHY PLL, indicated by the macro FSL_FEATURE_MIPI_DSI_NO_DPHY_PLL. For these platforms, the refClkFreq_Hz is useless.

Parameters

- base – MIPI DSI host peripheral base address.
- config – Pointer to the D-PHY configuration.
- refClkFreq_Hz – The REFCLK frequency in Hz.

Returns

The actual D-PHY PLL output frequency. If could not configure the PLL to the target frequency, the return value is 0.

void DSI_DeinitDphy(MIPI_DSI_HOST_Type *base)

Deinitializes the D-PHY.

Power down the D-PHY PLL and shut down D-PHY.

Parameters

- base – MIPI DSI host peripheral base address.

void DSI_GetDphyDefaultConfig(*dsi_dphy_config_t* *config, uint32_t txHsBitClk_Hz, uint32_t txEscClk_Hz)

Get the default D-PHY configuration.

Gets the default D-PHY configuration, the timing parameters are set according to D-PHY specification. User could use the configuration directly, or change some parameters according to the special device.

Parameters

- config – Pointer to the D-PHY configuration.
- txHsBitClk_Hz – High speed bit clock in Hz.
- txEscClk_Hz – Esc clock in Hz.

static inline void DSI_SetUlpsStatus(MIPI_DSI_HOST_Type *base, uint8_t ulpsStatus)

Sets the ULPS status for one clock lane and 4 data lanes.

Use status in _dsi_ulps_status as OR'ed mask value for ulpsStatus. Release the lanes from ULPS state before transfer.

Parameters

- base – MIPI DSI host peripheral base address.
- ulpsStatus – The ULPS status to set to the bus.

static inline uint8_t DSI_GetUlpsStatus(MIPI_DSI_HOST_Type *base)

Gets the ULPS status for one clock lane and 4 data lanes.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The current ULPS status.

static inline void DSI_EnableInterrupts(MIPI_DSI_HOST_Type *base, uint32_t intGroup1, uint32_t intGroup2)

Enable the interrupts.

The interrupts to enable are passed in as OR'ed mask value of _dsi_interrupt.

Parameters

- base – MIPI DSI host peripheral base address.
- intGroup1 – Interrupts to enable in group 1.

- intGroup2 – Interrupts to enable in group 2.

static inline void DSI_DisableInterrupts(MIPI_DSI_HOST_Type *base, uint32_t intGroup1, uint32_t intGroup2)

Disable the interrupts.

The interrupts to disable are passed in as OR'ed mask value of _dsi_interrupt.

Parameters

- base – MIPI DSI host peripheral base address.
- intGroup1 – Interrupts to disable in group 1.
- intGroup2 – Interrupts to disable in group 2.

static inline void DSI_GetAndClearInterruptStatus(MIPI_DSI_HOST_Type *base, uint32_t *intGroup1, uint32_t *intGroup2)

Get and clear the interrupt status.

Parameters

- base – MIPI DSI host peripheral base address.
- intGroup1 – Group 1 interrupt status.
- intGroup2 – Group 2 interrupt status.

static inline void DSI_SetDbiPixelFormat(MIPI_DSI_HOST_Type *base, *dsi_dbi_pixel_format_t* format)

brief Configure the DBI pixel format.

param base MIPI DSI host peripheral base address. param format of the pixel.

static inline void DSI_SetDbiPixelFifoSendLevel(MIPI_DSI_HOST_Type *base, uint16_t sendLevel)

Configure the DBI pixel FIFO send level.

This controls the level at which the DBI Host bridge begins sending pixels

Parameters

- base – MIPI DSI host peripheral base address.
- sendLevel – Send level value set to register.

static inline void DSI_SetDbiPixelPayloadSize(MIPI_DSI_HOST_Type *base, uint16_t payloadSize)

Configure the DBI pixel payload size.

Configures maximum number of pixels that should be sent as one DSI packet. Recommended to be evenly divisible by the line size (in pixels).

Parameters

- base – MIPI DSI host peripheral base address.
- payloadSize – Payload size value set to register.

void DSI_SetApbPacketControl(MIPI_DSI_HOST_Type *base, uint16_t wordCount, uint8_t virtualChannel, *dsi_tx_data_type_t* dataType, uint8_t flags)

Configure the APB packet to send.

This function configures the next APB packet transfer. After configuration, the packet transfer could be started with function *DSI_SendApbPacket*. If the packet is long packet, Use *DSI_WriteApbTxPayload* to fill the payload before start transfer.

Parameters

- base – MIPI DSI host peripheral base address.

- wordCount – For long packet, this is the byte count of the payload. For short packet, this is (data1 « 8) | data0.
- virtualChannel – Virtual channel.
- dataType – The packet data type, (DI).
- flags – The transfer control flags, see _dsi_transfer_flags.

void DSI_WriteApbTxPayload(MIPI_DSI_HOST_Type *base, const uint8_t *payload, uint16_t payloadSize)

Fill the long APB packet payload.

Write the long packet payload to TX FIFO.

Parameters

- base – MIPI DSI host peripheral base address.
- payload – Pointer to the payload.
- payloadSize – Payload size in byte.

void DSI_WriteApbTxPayloadExt(MIPI_DSI_HOST_Type *base, const uint8_t *payload, uint16_t payloadSize, bool sendDcsCmd, uint8_t dcsCmd)

Extended function to fill the payload to TX FIFO.

Write the long packet payload to TX FIFO. This function could be used in two ways

a. Include the DCS command in parameter payload. In this case, the DCS command is the first byte of payload. The parameter sendDcsCmd is set to false, the dcsCmd is not used. This function is the same as *DSI_WriteApbTxPayload* when used in this way.
b. The DCS command in not in parameter payload, but specified by parameter dcsCmd. In this case, the parameter sendDcsCmd is set to true, the dcsCmd is the DCS command to send. The payload is sent after dcsCmd.

Parameters

- base – MIPI DSI host peripheral base address.
- payload – Pointer to the payload.
- payloadSize – Payload size in byte.
- sendDcsCmd – If set to true, the DCS command is specified by dcsCmd, otherwise the DCS command is included in the payload.
- dcsCmd – The DCS command to send, only used when sendDCSCmd is true.

void DSI_ReadApbRxPayload(MIPI_DSI_HOST_Type *base, uint8_t *payload, uint16_t payloadSize)

Read the long APB packet payload.

Read the long packet payload from RX FIFO. This function reads directly but does not check the RX FIFO status. Upper layer should make sure there are available data.

Parameters

- base – MIPI DSI host peripheral base address.
- payload – Pointer to the payload.
- payloadSize – Payload size in byte.

static inline void DSI_SendApbPacket(MIPI_DSI_HOST_Type *base)

Trigger the controller to send out APB packet.

Send the packet set by *DSI_SetApbPacketControl*.

Parameters

- base – MIPI DSI host peripheral base address.

static inline uint32_t DSI_GetApbStatus(MIPI_DSI_HOST_Type *base)

Get the APB status.

The return value is OR'ed value of _dsi_apb_status.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The APB status.

static inline uint32_t DSI_GetRxErrorStatus(MIPI_DSI_HOST_Type *base)

Get the error status during data transfer.

The return value is OR'ed value of _dsi_rx_error_status.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The error status.

static inline uint8_t DSI_GetEccRxErrorPosition(uint32_t rxErrorStatus)

Get the one-bit RX ECC error position.

When one-bit ECC RX error detected using *DSI_GetRxErrorStatus*, this function could be used to get the error bit position.

```
uint8_t eccErrorPos;
uint32_t rxErrorStatus = DSI_GetRxErrorStatus(MIPI_DSI);
if (kDSI_RxErrorEccOneBit & rxErrorStatus)
{
    eccErrorPos = DSI_GetEccRxErrorPosition(rxErrorStatus);
}
```

Parameters

- rxErrorStatus – The error status returned by *DSI_GetRxErrorStatus*.

Returns

The 1-bit ECC error position.

static inline uint32_t DSI_GetAndClearHostStatus(MIPI_DSI_HOST_Type *base)

Get and clear the DSI host status.

The host status are returned as mask value of *_dsi_host_status*.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The DSI host status.

static inline uint32_t DSI_GetRxPacketHeader(MIPI_DSI_HOST_Type *base)

Get the RX packet header.

Parameters

- base – MIPI DSI host peripheral base address.

Returns

The RX packet header.

static inline *dsi_rx_data_type_t* DSI_GetRxPacketType(uint32_t rxPktHeader)

Extract the RX packet type from the packet header.

Extract the RX packet type from the packet header get by *DSI_GetRxPacketHeader*.

Parameters

- rxPktHeader – The RX packet header get by *DSI_GetRxPacketHeader*.

Returns

The RX packet type.

static inline uint16_t DSI_GetRxPacketWordCount(uint32_t rxPktHeader)

Extract the RX packet word count from the packet header.

Extract the RX packet word count from the packet header get by *DSI_GetRxPacketHeader*.

Parameters

- rxPktHeader – The RX packet header get by *DSI_GetRxPacketHeader*.

Returns

For long packet, return the payload word count (byte). For short packet, return the (data0 « 8) | data1.

static inline uint8_t DSI_GetRxPacketVirtualChannel(uint32_t rxPktHeader)

Extract the RX packet virtual channel from the packet header.

Extract the RX packet virtual channel from the packet header get by *DSI_GetRxPacketHeader*.

Parameters

- rxPktHeader – The RX packet header get by *DSI_GetRxPacketHeader*.

Returns

The virtual channel.

status_t DSI_TransferBlocking(MIPI_DSI_HOST_Type *base, *dsi_transfer_t* *xfer)

APB data transfer using blocking method.

Perform APB data transfer using blocking method. This function waits until all data send or received, or timeout happens.

When using this API to read data, the actually read data count could be got from xfer->rxDataSize.

Parameters

- base – MIPI DSI host peripheral base address.
- xfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data transfer finished with no error.
- kStatus_Timeout – Transfer failed because of timeout.
- kStatus_DSI_RxDataError – RX data error, user could use *DSI_GetRxErrorStatus* to check the error details.

- kStatus_DSI_ErrorReportReceived – Error Report packet received, user could use *DSI_GetAndClearHostStatus* to check the error report status.
- kStatus_DSI_NotSupported – Transfer format not supported.
- kStatus_DSI_Fail – Transfer failed for other reasons.

status_t DSI_TransferCreateHandle(MIPI_DSI_HOST_Type *base, *dsi_handle_t* *handle, *dsi_callback_t* callback, void *userData)

Create the MIPI DSI handle.

This function initializes the MIPI DSI handle which can be used for other transactional APIs.

Parameters

- base – MIPI DSI host peripheral base address.
- handle – Handle pointer.
- callback – Callback function.
- userData – User data.

status_t DSI_TransferNonBlocking(MIPI_DSI_HOST_Type *base, *dsi_handle_t* *handle, *dsi_transfer_t* *xfer)

APB data transfer using interrupt method.

Perform APB data transfer using interrupt method, when transfer finished, upper layer could be informed through callback function.

When using this API to read data, the actually read data count could be got from handle->xfer->rxDataSize after read finished.

Parameters

- base – MIPI DSI host peripheral base address.
- handle – pointer to dsi_handle_t structure which stores the transfer state.
- xfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Data transfer started successfully.
- kStatus_DSI_Busy – Failed to start transfer because DSI is busy with pervious transfer.
- kStatus_DSI_NotSupported – Transfer format not supported.

void DSI_TransferAbort(MIPI_DSI_HOST_Type *base, *dsi_handle_t* *handle)

Abort current APB data transfer.

Parameters

- base – MIPI DSI host peripheral base address.
- handle – pointer to dsi_handle_t structure which stores the transfer state.

void DSI_TransferHandleIRQ(MIPI_DSI_HOST_Type *base, *dsi_handle_t* *handle)

Interrupt handler for the DSI.

Parameters

- base – MIPI DSI host peripheral base address.
- handle – pointer to dsi_handle_t structure which stores the transfer state.

FSL_MIPI_DSI_DRIVER_VERSION

Error codes for the MIPI DSI driver.

Values:

enumerator kStatus_DSI_Busy
DSI is busy.

enumerator kStatus_DSI_RxDataError
Read data error.

enumerator kStatus_DSI_ErrorReportReceived
Error report package received.

enumerator kStatus_DSI_NotSupported
The transfer type not supported.

enum _dsi_dbi_pixel_format
MIPI DBI pixel format.

Values:

enumerator kDSI_DbiNoMapping
No pixel to bit mapping.

enumerator kDSI_DbiRGB888
Send data to interface as RGB888 format.

enumerator kDSI_DbiRGB666
Send data to interface as RGB666 format.

enumerator kDSI_DbiRGB565
Send data to interface as RGB565 format.

enumerator kDSI_DbiRGB444
Send data to interface as RGB444 format.

enumerator kDSI_DbiRGB332
Send data to interface as RGB332 format.

enum _dsi_dpi_color_coding
MIPI DPI interface color coding.

Values:

enumerator kDSI_Dpi16BitConfig1
16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_GGGBBBBB.

enumerator kDSI_Dpi16BitConfig2
16-bit configuration 2. RGB565: XXXRRRRR_XXGGGGGG_XXXBBBBB.

enumerator kDSI_Dpi16BitConfig3
16-bit configuration 3. RGB565: XXRRRRRX_XXGGGGGG_XXBBBBBX.

enumerator kDSI_Dpi18BitConfig1
18-bit configuration 1. RGB666: XXXXXXRR_RRRRGGGG_GGBBBBBB.

enumerator kDSI_Dpi18BitConfig2
18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBBBBB.

enumerator kDSI_Dpi24Bit
24-bit.

enum _dsi_dpi_pixel_packet
MIPI DSI pixel packet type send through DPI interface.
Values:

enumerator kDSI_PixelPacket16Bit
16 bit RGB565.

enumerator kDSI_PixelPacket18Bit
18 bit RGB666 packed.

enumerator kDSI_PixelPacket18BitLoosely
18 bit RGB666 loosely packed into three bytes.

enumerator kDSI_PixelPacket24Bit
24 bit RGB888, each pixel uses three bytes.

_dsi_dpi_polarity_flag DPI signal polarity.
Values:

enumerator kDSI_DpiVsyncActiveLow
VSYNC active low.

enumerator kDSI_DpiHsyncActiveLow
HSYNC active low.

enumerator kDSI_DpiVsyncActiveHigh
VSYNC active high.

enumerator kDSI_DpiHsyncActiveHigh
HSYNC active high.

enum _dsi_dpi_video_mode
DPI video mode.
Values:

enumerator kDSI_DpiNonBurstWithSyncPulse
Non-Burst mode with Sync Pulses.

enumerator kDSI_DpiNonBurstWithSyncEvent
Non-Burst mode with Sync Events.

enumerator kDSI_DpiBurst
Burst mode.

enum _dsi_dpi_bllp_mode
Behavior in BLLP (Blanking or Low-Power Interval).
Values:

enumerator kDSI_DpiBllpLowPower
LP mode used in BLLP periods.

enumerator kDSI_DpiBllpBlanking
Blanking packets used in BLLP periods.

enumerator kDSI_DpiBllpNull
Null packets used in BLLP periods.

_dsi_apb_status Status of APB to packet interface.
Values:

enumerator kDSI_ApbNotIdle
State machine not idle

enumerator kDSI_ApbTxDone
Tx packet done

enumerator kDSI_ApbRxControl
DPHY direction 0 - tx had control, 1 - rx has control

enumerator kDSI_ApbTxOverflow
TX fifo overflow

enumerator kDSI_ApbTxUnderflow
TX fifo underflow

enumerator kDSI_ApbRxOverflow
RX fifo overflow

enumerator kDSI_ApbRxUnderflow
RX fifo underflow

enumerator kDSI_ApbRxHeaderReceived
RX packet header has been received

enumerator kDSI_ApbRxPacketReceived
All RX packet payload data has been received

_dsi_rx_error_status Host receive error status.

Values:

enumerator kDSI_RxErrorEccOneBit
ECC single bit error detected.

enumerator kDSI_RxErrorEccMultiBit
ECC multi bit error detected.

enumerator kDSI_RxErrorCrc
CRC error detected.

enumerator kDSI_RxErrorHtxTo
High Speed forward TX timeout detected.

enumerator kDSI_RxErrorLrxTo
Reverse Low power data receive timeout detected.

enumerator kDSI_RxErrorBtaTo
BTA timeout detected.

enum __dsi_host_status
DSI host controller status (status_out)

Values:

enumerator kDSI_HostSoTError
SoT error from peripheral error report.

enumerator kDSI_HostSoTSyncError
SoT Sync error from peripheral error report.

enumerator kDSI_HostEoTSyncError
EoT Sync error from peripheral error report.

enumerator kDSI_HostEscEntryCmdError
Escape Mode Entry Command Error from peripheral error report.

enumerator kDSI_HostLpTxSyncError
Low-power transmit Sync Error from peripheral error report.

enumerator kDSI_HostPeriphToError
Peripheral timeout error from peripheral error report.

enumerator kDSI_HostFalseControlError
False control error from peripheral error report.

enumerator kDSI_HostContentionDetected
Contention detected from peripheral error report.

enumerator kDSI_HostEccErrorOneBit
Single bit ECC error (corrected) from peripheral error report.

enumerator kDSI_HostEccErrorMultiBit
Multi bit ECC error (not corrected) from peripheral error report.

enumerator kDSI_HostChecksumError
Checksum error from peripheral error report.

enumerator kDSI_HostInvalidDataType
DSI data type not recognized.

enumerator kDSI_HostInvalidVcId
DSI VC ID invalid.

enumerator kDSI_HostInvalidTxLength
Invalid transmission length.

enumerator kDSI_HostProtocalViolation
DSI protocal violation.

enumerator kDSI_HostResetTriggerReceived
Reset trigger received.

enumerator kDSI_HostTearTriggerReceived
Tear effect trigger receive.

enumerator kDSI_HostAckTriggerReceived
Acknowledge trigger message received.

_dsi_ulps_status Status of APB to packet interface.
Values:

enumerator kDSI_ClockLaneUlpsEnable
Clock lane in ULPS mode.

enumerator kDSI_DataLane0UlpsEnable
Data lane 0 in ULPS mode.

enumerator kDSI_DataLane1UlpsEnable
Data lane 1 in ULPS mode.

enumerator kDSI_DataLane2UlpsEnable
Data lane 2 in ULPS mode.

enumerator kDSI_DataLane3UlpsEnable
Data lane 3 in ULPS mode.

_dsi_interrupt DSI interrupt.
Values:

enumerator kDSI_InterruptGroup1ApbNotIdle
State machine not idle

enumerator kDSI_InterruptGroup1ApbTxDone
Tx packet done

enumerator kDSI_InterruptGroup1ApbRxControl
DPHY direction 0 - tx control, 1 - rx control

enumerator kDSI_InterruptGroup1ApbTxOverflow
TX fifo overflow

enumerator kDSI_InterruptGroup1ApbTxUnderflow
TX fifo underflow

enumerator kDSI_InterruptGroup1ApbRxOverflow
RX fifo overflow

enumerator kDSI_InterruptGroup1ApbRxUnderflow
RX fifo underflow

enumerator kDSI_InterruptGroup1ApbRxHeaderReceived
RX packet header has been received

enumerator kDSI_InterruptGroup1ApbRxPacketReceived
All RX packet payload data has been received

enumerator kDSI_InterruptGroup1SoTError
SoT error from peripheral error report.

enumerator kDSI_InterruptGroup1SoTSyncError
SoT Sync error from peripheral error report.

enumerator kDSI_InterruptGroup1EoTSyncError
EoT Sync error from peripheral error report.

enumerator kDSI_InterruptGroup1EscEntryCmdError
Escape Mode Entry Command Error from peripheral error report.

enumerator kDSI_InterruptGroup1LpTxSyncError
Low-power transmit Sync Error from peripheral error report.

enumerator kDSI_InterruptGroup1PeriphToError
Peripheral timeout error from peripheral error report.

enumerator kDSI_InterruptGroup1FalseControlError
False control error from peripheral error report.

enumerator kDSI_InterruptGroup1ContentionDetected
Contention detected from peripheral error report.

enumerator kDSI_InterruptGroup1EccErrorOneBit
Single bit ECC error (corrected) from peripheral error report.

enumerator kDSI_InterruptGroup1EccErrorMultiBit
Multi bit ECC error (not corrected) from peripheral error report.

enumerator kDSI_InterruptGroup1ChecksumError
Checksum error from peripheral error report.

enumerator kDSI_InterruptGroup1InvalidDataType
DSI data type not recognized.

enumerator kDSI_InterruptGroup1InvalidVcId
DSI VC ID invalid.

enumerator kDSI_InterruptGroup1InvalidTxLength
Invalid transmission length.

enumerator kDSI_InterruptGroup1ProtocalViolation
DSI protocal violation.

enumerator kDSI_InterruptGroup1ResetTriggerReceived
Reset trigger received.

enumerator kDSI_InterruptGroup1TearTriggerReceived
Tear effect trigger receive.

enumerator kDSI_InterruptGroup1AckTriggerReceived
Acknowledge trigger message received.

enumerator kDSI_InterruptGroup1HtxTo
High speed TX timeout.

enumerator kDSI_InterruptGroup1LrxTo
Low power RX timeout.

enumerator kDSI_InterruptGroup1BtaTo
Host BTA timeout.

enumerator kDSI_InterruptGroup2EccOneBit
Sinle bit ECC error.

enumerator kDSI_InterruptGroup2EccMultiBit
Multi bit ECC error.

enumerator kDSI_InterruptGroup2CrcError
CRC error.

enum _dsi_tx_data_type
DSI TX data type.
Values:

enumerator kDSI_TxDataVsyncStart
V Sync start.

enumerator kDSI_TxDataVsyncEnd
V Sync end.

enumerator kDSI_TxDataHsyncStart
H Sync start.

enumerator kDSI_TxDataHsyncEnd
H Sync end.

enumerator kDSI_TxDataEoTp
End of transmission packet.

enumerator kDSI_TxDataCmOff
Color mode off.

enumerator kDSI_TxDataCmOn
Color mode on.

enumerator kDSI_TxDataShutDownPeriph
Shut down peripheral.

enumerator kDSI_TxDataTurnOnPeriph
Turn on peripheral.

enumerator kDSI_TxDataGenShortWrNoParam
Generic Short WRITE, no parameters.

enumerator kDSI_TxDataGenShortWrOneParam
Generic Short WRITE, one parameter.

enumerator kDSI_TxDataGenShortWrTwoParam
Generic Short WRITE, two parameter.

enumerator kDSI_TxDataGenShortRdNoParam
Generic Short READ, no parameters.

enumerator kDSI_TxDataGenShortRdOneParam
Generic Short READ, one parameter.

enumerator kDSI_TxDataGenShortRdTwoParam
Generic Short READ, two parameter.

enumerator kDSI_TxDataDcsShortWrNoParam
DCS Short WRITE, no parameters.

enumerator kDSI_TxDataDcsShortWrOneParam
DCS Short WRITE, one parameter.

enumerator kDSI_TxDataDcsShortRdNoParam
DCS Short READ, no parameters.

enumerator kDSI_TxDataSetMaxReturnPktSize
Set the Maximum Return Packet Size.

enumerator kDSI_TxDataNull
Null Packet, no data.

enumerator kDSI_TxDataBlanking
Blanking Packet, no data.

enumerator kDSI_TxDataGenLongWr
Generic long write.

enumerator kDSI_TxDataDcsLongWr
DCS Long Write/write_LUT Command Packet.

enumerator kDSI_TxDataLooselyPackedPixel20BitYCbCr
Loosely Packed Pixel Stream, 20-bit YCbCr, 4:2:2 Format.

enumerator kDSI_TxDataPackedPixel24BitYCbCr
Packed Pixel Stream, 24-bit YCbCr, 4:2:2 Format.

enumerator kDSI_TxDataPackedPixel16BitYCbCr
Packed Pixel Stream, 16-bit YCbCr, 4:2:2 Format.

enumerator kDSI_TxDataPackedPixel30BitRGB
Packed Pixel Stream, 30-bit RGB, 10-10-10 Format.

enumerator kDSI_TxDataPackedPixel36BitRGB
Packed Pixel Stream, 36-bit RGB, 12-12-12 Format.

enumerator kDSI_TxDataPackedPixel12BitYCrCb
Packed Pixel Stream, 12-bit YCbCr, 4:2:0 Format.

enumerator kDSI_TxDataPackedPixel16BitRGB
Packed Pixel Stream, 16-bit RGB, 5-6-5 Format.

enumerator kDSI_TxDataPackedPixel18BitRGB
Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.

enumerator kDSI_TxDataLooselyPackedPixel18BitRGB
Loosely Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.

enumerator kDSI_TxDataPackedPixel24BitRGB
Packed Pixel Stream, 24-bit RGB, 8-8-8 Format.

enum _dsi_rx_data_type
DSI RX data type.
Values:

enumerator kDSI_RxDataAckAndErrorReport
Acknowledge and Error Report

enumerator kDSI_RxDataEoTp
End of Transmission packet.

enumerator kDSI_RxDataGenShortRdResponseOneByte
Generic Short READ Response, 1 byte returned.

enumerator kDSI_RxDataGenShortRdResponseTwoByte
Generic Short READ Response, 2 byte returned.

enumerator kDSI_RxDataGenLongRdResponse
Generic Long READ Response.

enumerator kDSI_RxDataDcsLongRdResponse
DCS Long READ Response.

enumerator kDSI_RxDataDcsShortRdResponseOneByte
DCS Short READ Response, 1 byte returned.

enumerator kDSI_RxDataDcsShortRdResponseTwoByte
DCS Short READ Response, 2 byte returned.

_dsi_transfer_flags DSI transfer control flags.

Values:

enumerator kDSI_TransferUseHighSpeed
Use high speed mode or not.

enumerator kDSI_TransferPerformBTA
Perform BTA or not.

typedef enum *_dsi_dbi_pixel_format* dsi_dbi_pixel_format_t
MIPI DBI pixel format.

typedef struct *_dsi_config* dsi_config_t
MIPI DSI controller configuration.

typedef enum *_dsi_dpi_color_coding* dsi_dpi_color_coding_t
MIPI DPI interface color coding.

typedef enum *_dsi_dpi_pixel_packet* dsi_dpi_pixel_packet_t
MIPI DSI pixel packet type send through DPI interface.

typedef enum *_dsi_dpi_video_mode* dsi_dpi_video_mode_t
DPI video mode.

typedef enum *_dsi_dpi_bllp_mode* dsi_dpi_bllp_mode_t
Behavior in BLLP (Blanking or Low-Power Interval).

typedef struct *_dsi_dpi_config* dsi_dpi_config_t
MIPI DSI controller DPI interface configuration.

typedef struct *_dsi_dphy_config* dsi_dphy_config_t
MIPI DSI D-PHY configuration.

typedef enum *_dsi_tx_data_type* dsi_tx_data_type_t
DSI TX data type.

typedef enum *_dsi_rx_data_type* dsi_rx_data_type_t
DSI RX data type.

typedef struct *_dsi_transfer* dsi_transfer_t
Structure for the data transfer.

typedef struct *_dsi_handle* dsi_handle_t
MIPI DSI transfer handle.

typedef void (*dsi_callback_t)(MIPI_DSI_HOST_Type *base, *dsi_handle_t* *handle, *status_t* status, void *userData)

MIPI DSI callback for finished transfer.

When transfer finished, one of these status values will be passed to the user:

- *kStatus_Success* Data transfer finished with no error.
- *kStatus_Timeout* Transfer failed because of timeout.
- *kStatus_DSI_RxDataError* RX data error, user could use *DSI_GetRxErrorStatus* to check the error details.
- *kStatus_DSI_ErrorReportReceived* Error Report packet received, user could use *DSI_GetAndClearHostStatus* to check the error report status.
- *kStatus_Fail* Transfer failed for other reasons.

FSL_DSI_TX_MAX_PAYLOAD_BYTE

FSL_DSI_RX_MAX_PAYLOAD_BYTE

struct _dsi_config

#include <fsl_mipi_dsi.h> MIPI DSI controller configuration.

Public Members

uint8_t numLanes

Number of lanes.

bool enableNonContinuousHsClk

In enabled, the high speed clock will enter low power mode between transmissions.

bool enableTxUlps

Enable the TX ULPS.

bool autoInsertEoTp

Insert an EoTp short package when switching from HS to LP.

uint8_t numExtraEoTp

How many extra EoTp to send after the end of a packet.

uint32_t htxTo_ByteClk

HS TX timeout count (HTX_TO) in byte clock.

uint32_t lrxHostTo_ByteClk

LP RX host timeout count (LRX-H_TO) in byte clock.

uint32_t btaTo_ByteClk

Bus turn around timeout count (TA_TO) in byte clock.

struct _dsi_dpi_config

#include <fsl_mipi_dsi.h> MIPI DSI controller DPI interface configuration.

Public Members

uint16_t pixelPayloadSize

Maximum number of pixels that should be sent as one DSI packet. Recommended that the line size (in pixels) is evenly divisible by this parameter.

dsi_dpi_color_coding_t dpiColorCoding

DPI color coding.

dsi_dpi_pixel_packet_t pixelPacket

Pixel packet format.

dsi_dpi_video_mode_t videoMode

Video mode.

dsi_dpi_bllp_mode_t bllpMode

Behavior in BLLP.

uint8_t polarityFlags

OR'ed value of _dsi_dpi_polarity_flag controls signal polarity.

uint16_t hfp

Horizontal front porch, in dpi pixel clock.

uint16_t hbp

Horizontal back porch, in dpi pixel clock.

uint16_t hsw

Horizontal sync width, in dpi pixel clock.

uint8_t vfp

Number of lines in vertical front porch.

uint8_t vbp

Number of lines in vertical back porch.

uint16_t panelHeight

Line number in vertical active area.

uint8_t virtualChannel

Virtual channel.

struct __dsi__dphy__config

#include <fsl_mipi_dsi.h> MIPI DSI D-PHY configuration.

Public Members

uint32_t txHsBitClk_Hz

The generated HS TX bit clock in Hz.

uint8_t tClkPre_ByteClk

TLPX + TCLK-PREPARE + TCLK-ZERO + TCLK-PRE in byte clock. Set how long the controller will wait after enabling clock lane for HS before enabling data lanes for HS.

uint8_t tClkPost_ByteClk

TCLK-POST + T_CLK-TRAIL in byte clock. Set how long the controller will wait before putting clock lane into LP mode after data lanes detected in stop state.

uint8_t tHsExit_ByteClk

THS-EXIT in byte clock. Set how long the controller will wait after the clock lane has been put into LP mode before enabling clock lane for HS again.

uint32_t tWakeup_EscClk

Number of clk_esc clock periods to keep a clock or data lane in Mark-1 state after exiting ULPS.

uint8_t tHsPrepare_HalfEscClk

THS-PREPARE in clk_esc/2. Set how long to drive the LP-00 state before HS transmissions, available values are 2, 3, 4, 5.

uint8_t tClkPrepare_HalfEscClk

TCLK-PREPARE in clk_esc/2. Set how long to drive the LP-00 state before HS transmissions, available values are 2, 3.

uint8_t tHsZero_ByteClk

THS-ZERO in clk_byte. Set how long that controller drives data lane HS-0 state before transmit the Sync sequence. Available values are 6, 7, ..., 37.

uint8_t tClkZero_ByteClk

TCLK-ZERO in clk_byte. Set how long that controller drives clock lane HS-0 state before transmit the Sync sequence. Available values are 3, 4, ..., 66.

uint8_t tHsTrail_ByteClk

THS-TRAIL + 4*UI in clk_byte. Set the time of the flipped differential state after last payload data bit of HS transmission burst. Available values are 0, 1, ..., 15.

uint8_t tClkTrail_ByteClk

TCLK-TRAIL + 4*UI in clk_byte. Set the time of the flipped differential state after last payload data bit of HS transmission burst. Available values are 0, 1, ..., 15.

struct _dsi_transfer

#include <fsl_mipi_dsi.h> Structure for the data transfer.

Public Members

uint8_t virtualChannel
: Virtual channel.

dsi_tx_data_type_t txDataType
: TX data type.

uint8_t flags
: Flags to control the transfer, see _dsi_transfer_flags.

const uint8_t *txData
: The TX data buffer.

uint8_t *rxData
: The RX data buffer.

uint16_t txDataSize
: Size of the TX data.

uint16_t rxDataSize
: Size of the RX data.

bool sendDcsCmd
: If set to true, the DCS command is specified by *dcsCmd*, otherwise the DCS command is included in the *txData*.

uint8_t dcsCmd
: The DCS command to send, only valid when *sendDcsCmd* is true.

struct __dsi__handle
: *#include <fsl_mipi_dsi.h>* MIPI DSI transfer handle structure.

Public Members

volatile bool isBusy
: MIPI DSI is busy with APB data transfer.

dsi_transfer_t xfer
: Transfer information.

dsi_callback_t callback
: DSI callback

void *userData
: Callback parameter

2.61 MIPI_DSI: MIPI DSI Host Controller

2.62 MMU: Memory Map Unit

void MMU_Init(MMU_Type *base, const *mmu_config_t* *config)

Initialize MMU:

a. Enable MMU clock;
b. Define the basic unit that is used in the memory map;
c. Configure valid address range;
d. Fill LUT SRAM;
e. Enable MMU remap function.

Parameters

- base – MMU peripheral base address.
- config – Pointer to the user defined variable in type of *mmu_config_t*.

void MMU_Deinit(MMU_Type *base)

De-initialize MMU:

a. Disable MMU remap function;
b. Reset MMU logic;
c. Disable MMU clock.

Parameters

- base –

void MMU_GetDefaultConfig(*mmu_config_t* *config)

Get default configurations of MMU module.

```
config->unitSize = kMMU_UnitSize4KB;
config->pageMode = kMMU_1PageMode;
config->invalidAddrMode = kMMU_InvalidAddrRespondError;
config->baseAddr    = 0x0UL;
config->maxAddr     = 0x0UL;
config->operationToWrite = kMMU_LUTBurstWrite;
config->ptrLutWord       = NULL;
config->countOfWordToWrite = 0U;
config->startPointer = 0U;
```

Parameters

- config – Pointer to the user defined variable in type of *mmu_config_t*.

void MMU_SetupSimpleMap(MMU_Type *base, uint32_t virt, uint32_t phys)

Map one physical unit to the virtual unit.

Parameters

- base – MMU peripheral base address.
- virt – The unit-aligned virtual address.

- phys – The unit-aligned virtual address.

void MMU_TearDownSimpleMap(MMU_Type *base, uint32_t virt)

Remove mapping for one unit of provided virtual address.

Parameters

- base – MMU peripheral base address.
- virt – The unit-aligned virtual address.

void MMU_SetupRegionMap(MMU_Type *base, uint32_t virt, uint32_t phys, size_t regionSize)

Map a region of physical memory into the virtual address space.

Parameters

- base – MMU peripheral base address.
- virt – The unit-aligned destination virtual address to map.
- phys – The unit-aligned source physical address to map.
- regionSize – Unit-aligned size of mapped memory region in bytes.

void MMU_TearDownRegionMap(MMU_Type *base, uint32_t virt, size_t regionSize)

Remove mapping for a provided virtual address region.

Parameters

- base – MMU peripheral base address.
- virt – The unit-aligned virtual address.
- regionSize – The unit-aligned size of region in bytes.

void MMU_RemapRegion(MMU_Type *base, uint32_t oldVirt, uint32_t newVirt, size_t regionSize)

Remap old virtual region into new address.

Parameters

- base – MMU peripheral base address.
- oldVirt – The unit-aligned old virtual address.
- newVirt – The unit-aligned new virtual address.
- regionSize – The unit-aligned size of region in bytes.

void MMU_MoveRegion(MMU_Type *base, uint32_t oldVirt, uint32_t newVirt, uint32_t newPhys, size_t regionSize)

Physically move region of memory, with copy.

Maps a region of physical memory into the new virtual address space, and copy region of size from the old virtual address space. The new virtual memory region is mapped from physical memory starting newPhys of size regionSize.

Parameters

- base – MMU peripheral base address.
- oldVirt – Unit-aligned old virtual region address.
- newVirt – Unit-aligned new virtual region address.
- newPhys – Unit-aligned new physical region address which will map to new virtual region address.
- regionSize – Unit-aligned size of region in bytes.

void MMU_SetupArrayMap(MMU_Type *base, uint32_t *ptrVirts, uint32_t *ptrPhys, uint32_t count)

Map an array of physical memory address into an array of virtual memory address.

Parameters

- base – MMU peripheral base address.
- ptrVirts – Pointer to the array which contain unit-aligned virtual address.
- ptrPhys – Pointer to the array which contain unit-aligned physical address.
- count – Count of array elements.

void MMU_TearDownArrayMap(MMU_Type *base, uint32_t *ptrVirts, uint32_t count)

Remove mapping of an array of virtual address.

Parameters

- base – MMU peripheral base address.
- ptrVirts – Pointer to the array which contains unit-aligned virtual address.
- count – Count of array elements.

void MMU_MoveArray(MMU_Type *base, uint32_t *ptrOldVirts, uint32_t *ptrNewVirts, uint32_t *ptrNewPhys, uint32_t count)

Physical move array of memory, with copy.

Maps an array of physical memory into the new array of virtual address space(ptrNewVirts), and copy an unit size of data from old array of virtual address to new array of virtual address.

Parameters

- base – MMU peripheral base address.
- ptrOldVirts – Pointer to the array which contain old unit-aligned virtual address.
- ptrNewVirts – Pointer to the array which contain new unit-aligned virtual address.
- ptrNewPhys – Pointer to the array which contain new unit-aligned physical address.
- count – Count of array elements.

void MMU_GetPhyAddr(MMU_Type *base, uint32_t virt, uint32_t *ptrPtrs)

Get mapped physical address from input virtual address.

Parameters

- base – MMU peripheral base address.
- virt – The virtual address.
- ptrPtrs – **[out]** Pointer to the variable to store mapped physical address.

static inline void MMU_EnableMemoryMap(MMU_Type *base, bool enable)

Enable/disable memory map.

Parameters

- base – MMU peripheral base address.
- enable – Used to enable/disable memory map:
  - **true** Enable memory map, the addresses of the AHB transaction are remapped by looking up LUT SRAM.

– **false** Disable memory map, the AHB transaction passes through MMU without address remapping.

static inline void MMU_UpdateUnitSize(MMU_Type *base, *mmu_unit_size_t* unitSize)

Update the basic size of the unit that is used in the memory map.

Parameters

- base – MMU peripheral base address.
- unitSize – Used to specify the basic size of unit in type of *mmu_unit_size_t*.

void MMU_DoSoftwareReset(MMU_Type *base)

Trigger a software reset to reset the MMU logic.

Parameters

- base – MMU peripheral base address.

static inline void MMU_UpdateValidAddressRange(MMU_Type *base, uint16_t baseAddr, uint16_t maxAddr)

Update the valid address range.

```
To set AHB address from 0x51000000UL to 0x511FFFFF are valid:
MMU_UpdateValidAddressRange(MMU0, 0x5100U, 0x511FU);
```

Note: Both baseAddr and maxAddr mean the upper 16 bits of a valid address.

Parameters

- base – MMU peripheral base address.
- baseAddr – Upper 16 bits of valid range's base address.
- maxAddr – Upper 16 bits of valid range's maximum address.

static inline void MMU_ChangePageOption(MMU_Type *base, *mmu_page_option_t* pageOpt)

Change page mode, the MMU module supports #kMMU_1PageMode and #kMMU_4PageMode.

Parameters

- base – MMU peripheral base address.
- pageMode – Page mode to use, please refer to *mmu_page_option_t*.

void MMU_UpdateLUTBlocking(MMU_Type *base, uint16_t startPointer, *mmu_lut_word_t* *ptrLutWord, uint16_t countOfWords, *mmu_lut_write_operation_t* writeOperation)

Use the blocking method to update LUT SRAM with user-defined words.

Note: This function only return when all words that #ptrLutWord refers are set into LUT SRAM.

Note: This function supports both single write and burst write described in RM, please note that burst write is only useful when to fill continuous value into continuous addresses.

Note: startPointer + countOfWords should not exceed the maximum size of LUT SRAM(8192).

```
Fill continuous values into continuous addresses (E.g: initialize LUT SRAM address 2 and 3 with value␣
↪0x12 and 0x13)
        mmu_lut_word_t    startLutWord = {
            .mappedValue = 0x12U;
            .valid = 1U;
        };
        MMU_UpdateLUTBlocking(MMU0, 2U, &startLutWord, 2U, kMMU_LUTBurstWrite);
```

```
Fill discontinuous values into continuous addresses(E.g: initialize LUT SRAM address 2 and 3 with␣
↪value 0x12 and
0x34) mmu_lut_word_t lutWordArray[2] = {
            {
                .mappedValue = 0x12U;
                .valid       = 1U;
            },
            {
                .mappedValue = 0x34U;
                .valid       = 1U;
            },
        };
        MMU_UpdateLUTBlocking(MMU0, 2U, lutWordArray, 2U,kMMU_LUTSingleWrite);
```

```
Fill discontinuous values into discontinuous addresses(E.g: initialize LUT SRAM address 2 and 5 with␣
↪value 0x12 and
0x34)): mmu_lut_word_t addr2LutWord = { .mappedValue = 0x12U; .valid       = 1U;
        };
        MMU_UpdateLUTBlocking(MMU0, 2U, &addr2LutWord, 1U, kMMU_LUTSingleWrite);

        mmu_lut_word_t addr5LutWord = {
            .mappedValue = 0x34U;
            .valid       = 1U;
        };
        MMU_UpdateLUTBlocking(MMU0, 5U, &addr5LutWord, 1U, kMMU_LUTSingleWrite);
```

Parameters

- base – MMU peripheral base address.
- startPointer – Start pointer of LUT SRAM to fill new mapping words, range from 0 to 8191.
- ptrLutWord – Pointer to new lut words: if **writeOperation** is selected as **kMMU_LUTBurstWrite** this parameter should be a pointer to the variable which represents the first LUT word to set; if **writeOperation** is selected as **kMMU_LUTSingleWrite** this parameter can be set as a pointer to the array that contains all new LUT words to set or set as a pointer to one variable when only want to update one word into LUT's specific pointer.
- countOfWord – Count to new word to update into LUT SRAM, range from 1 to 8192.
- writeOperation – Specify the write operation, the meaning of #ptrLutWord is different for different write operations.

void MMU_ReadLUT(MMU_Type *base, uint16_t startPointer, *mmu_lut_word_t* *ptrWordReadBuffer, uint16_t count)

Read the specified number of LUT content from the specified pointer into the user-defined

buffer.

Note: #startPointer + #count should not exceed maximum size of LUT SRAM(8192).

Parameters

- base – MMU peripheral base address.
- startPointer – Specify start address to read in LUT SRAM, range from 0 to 8191.
- ptrWordReadBuffer – Pointer of a buffer that record read out words.
- count – Specify number of continuous address in LUT SRAM to read, range from 1 to 8192.

void MMU_TriggerLUTVerificationBlocking(MMU_Type *base, uint16_t startPointer, *mmu_lut_word_t* goldenWordToCheck, uint16_t count)

Trigger the specified number of read operations from LUT SRAM, compare with the input golden word, the read address and golden data are increased automatically each time.

Note: This function only return when all read-verify operations are complete.

Parameters

- base – MMU peripheral base address.
- startPointer – The pointer of LUT SRAM which start to check, range from 0 to 8191.
- goldenWordToCheck – The first golden word to check, in following read-verify operation, the golden data will increase automatically.
- count – Specify number of continuous address in LUT SRAM to check, range from 1 to 8192.

static inline void MMU_LockLUT(MMU_Type *base)

Lock LUT SRAM to avoid an accidental update.

Note: If locked, writing to LUT SRAM will trigger *kMMU_LockErrorFlag*.

Parameters

- base – MMU peripheral base address.

static inline void MMU_UnlockLUT(MMU_Type *base)

Unlock LUT SRAM.

Note: To update LUT SRAM, please ensure LUT is unlocked.

Parameters

- base – MMU peripheral base address.

static inline uint32_t MMU_GetStatusFlags(MMU_Type *base)

Get status flags.

Parameters

- base – MMU peripheral base address.

Returns

All status flags asserted, should be the OR'ed value of *mmu_status_flags_t*.

static inline void MMU_ClearStatusFlags(MMU_Type *base, uint32_t mask)

Clear status flags.

Parameters

- base – MMU peripheral base address.
- mask – The mask of flags to clear, should be the OR'ed value of *mmu_status_flags_t*.

static inline uint16_t MMU_GetIPSReadErrorAddress(MMU_Type *base)

Get the LUT SRAM address when an error is detected for the IPS read operation (including IPS read parity error and read-verify mismatch error).

Parameters

- base – MMU peripheral base address.

Returns

The pointer in LUT SRAM, range from 0 to 8191.

void MMU_GetAHBErrorRecord(MMU_Type *base, *mmu_ahb_error_record_t* *ptrRecord)

Get a record of AHB information when an error happens.

Parameters

- base – MMU peripheral base address.
- ptrRecord – Pointer to user-defined *mmu_ahb_error_record_t* variable which used to store current AHB error information.

Returns

The record of AHB information when an error happens, please refer to *mmu_ahb_error_record_t* for details.

static inline bool MMU_CheckMMUBusy(MMU_Type *base)

Check if MMU is in the Busy state.

Note: There are some possible reasons to make MMU is busy state:

- Initialization sequence
- Burst-write operation
- Read-verify operation

Parameters

- base – MMU peripheral base address.

Return values

- false – MMU not in the busy state.
- true – MMU is in the busy state, wait for it is not busy before taking an action.

static inline void MMU_EnableInterrupts(MMU_Type *base, uint32_t mask)

Enabled interrupts.

Parameters

- base – MMU peripheral base address.
- mask – Mask of interrupts to enable, should be the OR'ed value of *mmu_interrupt_enable_t*.

static inline void MMU_DisableInterrupts(MMU_Type *base, uint32_t mask)

Disable interrupts.

Parameters

- base – MMU peripheral base address.
- mask – Mask of interrupts to disable, should be the OR'ed value of *mmu_interrupt_enable_t*.

FSL_MMU_DRIVER_VERSION

mmu driver version 2.0.1.

enum _mmu_status_flags

The enumeration of MMU status flags,.

Values:

enumerator kMMU_BusyFlag

MMU is in Busy state, wait for it to clear before taking action.

enumerator kMMU_LUTInvalidFlag

Indicate whether the valid bit in the access address is 0, which means the access operation is invalid in LUT.

enumerator kMMU_AddressInvalidFlag

The AHB access address is out of the valid address range.

enumerator kMMU_ParityErrorFlag

A parity error is detected in the access address.

enumerator kMMU_LockErrorFlag

A lock error is detected when writing to LUT SRAM.

enumerator kMMU_IpsReadErrorFlag

Indicate whether an IPS read error is detected when read LUT SRAM, including IPS read parity error and read-verify mismatch error.

enumerator kMMU_AllFlags

All flags.

enum _mmu_interrupt_enable

The enumeration of MMU interrupts,.

Values:

enumerator kMMU_LUTInvalidInterruptEnable
Enable the interrupt for *kMMU_LUTInvalidFlag*.

enumerator kMMU_AddressInvalidInterruptEnable
Enable the interrupt for *kMMU_AddressInvalidFlag*.

enumerator kMMU_ParityErrorInterruptEnable
Enable the interrupt for *kMMU_ParityErrorFlag*.

enumerator kMMU_LockErrorInterruptEnable
Enable the interrupt for *kMMU_LockErrorFlag*.

enumerator kMMU_IpsReadErrorInterruptEnable
Enable the interrupt for *kMMU_IpsReadErrorFlag*.

enumerator kMMU_AllInterruptsEnable
Enable all interrupts.

enum _mmu_unit_size
The enumeration of unit size, available selections are 4KB, 8KB, 16KB, 32KB, and 64KB.
Values:

enumerator kMMU_UnitSize4KB
Specify the basic size of unit is 4KB.

enumerator kMMU_UnitSize8KB
Specify the basic size of unit is 8KB.

enumerator kMMU_UnitSize16KB
Specify the basic size of unit is 16KB.

enumerator kMMU_UnitSize32KB
Specify the basic size of unit is 32KB.

enumerator kMMU_UnitSize64KB
Specify the basic size of unit is 64KB.

enum _mmu_ahb_access_type
The enumeration of AHB access type, available selections are read access and write access.
Values:

enumerator kMMU_ReadAccess
The access type of the AHB transfer is read.

enumerator kMMU_WriteAccess
The access type of the AHB transfer is write.

enum _mmu_ahb_error_type

The enumeration of error type in case of AHB transfer, available selections are LUT invalid error, address invalid error, and parity error.

Values:

enumerator kMMU_LUTInvalidError

The error type information of the transfer is LUT invalid error.

enumerator kMMU_AddrInvalidError

The error type information of the transfer is address invalid error.

enumerator kMMU_ParityError

The error type information of the transfer is parity error.

enum _mmu_page_option

The MMU module supports two different page mode: 1 page mode and 4 pages mode.

Note: In 4 pages mode, each page contains 2K word. Different page can contain different mapping schema.

Values:

enumerator kMMU_PageOpt_1Page

MMU takes LUT SRAM as 1 page for address remapping.

enumerator kMMU_PageOpt_4Page_Low2KWordSelected

MMU takes LUT SRAM as 4 pages for address remapping, in this page mode the whole LUT is divided into 4 pages, use LUT SRAM low 2K words for address remapping.

enumerator kMMU_PageOpt_4Page_MidLow2KWordSelected

MMU takes LUT SRAM as 4 pages for address remapping, in this page mode the whole LUT is divided into 4 pages, use LUT SRAM mid-low 2K words for address remapping.

enumerator kMMU_PageOpt_4Page_MidHigh2KWordSelected

MMU takes LUT SRAM as 4 pages for address remapping, in this page mode the whole LUT is divided into 4 pages, use LUT SRAM mid-high 2K words for address remapping.

enumerator kMMU_PageOpt_4Page_High2KWordSelected

MMU takes LUT SRAM as 4 pages for address remapping, in this page mode the whole LUT is divided into 4 pages, use LUT SRAM mid-low 2K words for address remapping.

enum _mmu_lut_write_operation

The enumeration of LUT SRAM write operation, both burst-write and single-write are supported.

Values:

enumerator kMMU_LUTSingleWrite

In this operation, words to write in consequent LUT pointer may not consequent. It is possible to write discontinuous value into discontinuous address.

enumerator kMMU_LUTBurstWrite

In this operation, words to write in consequence LUT pointer must be consequent. write continuous value into continuous address.

enum _mmu_invalid_address_mode

The behavior when an invalid address is detected is programmable.

Values:

enumerator kMMU_InvalidAddrRespondError

MMU responds with a transfer error when a invalid address is detected.

enumerator kMMU_InvalidAddrAllowed

MMU allows transfer when an invalid address is detected.

typedef enum *_mmu_unit_size* mmu_unit_size_t

The enumeration of unit size, available selections are 4KB, 8KB, 16KB, 32KB, and 64KB.

typedef enum *_mmu_ahb_access_type* mmu_ahb_access_type_t

The enumeration of AHB access type, available selections are read access and write access.

typedef enum *_mmu_ahb_error_type* mmu_ahb_error_type_t

The enumeration of error type in case of AHB transfer, available selections are LUT invalid error, address invalid error, and parity error.

typedef enum *_mmu_page_option* mmu_page_option_t

The MMU module supports two different page mode: 1 page mode and 4 pages mode.

Note: In 4 pages mode, each page contains 2K word. Different page can contain different mapping schema.

typedef enum *_mmu_lut_write_operation* mmu_lut_write_operation_t

The enumeration of LUT SRAM write operation, both burst-write and single-write are supported.

typedef enum *_mmu_invalid_address_mode* mmu_invalid_address_mode_t

The behavior when an invalid address is detected is programmable.

typedef struct *_mmu_ahb_error_record* mmu_ahb_error_record_t

The record of AHB information when an error happens.

typedef struct *_mmu_lut_word* mmu_lut_word_t

The word in LUT SRAM, there are 8K words exist in LUT SRAM.

typedef struct *_mmu_config* mmu_config_t

The structure used to initialize the MMU instance includes unit size, page mode, address range, and LUT content.

mmu_unit_size_t g_selectedUnitSize

mmu_page_option_t g_selectedPageOpt

MMU_NUM_OF_UNIT_SIZE

MMU_PAGE_OPT_MASK

MMU_PAGE_OPT_SHIFT

MMU_PAGE_OPT(x)

struct _mmu_ahb_error_record

#include <fsl_mmu.h> The record of AHB information when an error happens.

Public Members

uint32_t masterId

Indicate the master ID of the transfer.

uint32_t protectionInfo

Indicate the protection information of the transfer.

mmu_ahb_access_type_t accessType

Indicate the access type of the transfer.

mmu_ahb_error_type_t errorType

Indicate the error type of the transfer.

uint32_t failAddr

Indicates the LUT address where an erro happens.

struct _mmu_lut_word

#include <fsl_mmu.h> The word in LUT SRAM, there are 8K words exist in LUT SRAM.

Public Members

uint16_t mappedValue

The mapped value to set, should be index of unit in valid address range not specific AHB address.

uint16_t valid

Specify whether the mapped value is valid: 1b1 means valid, 1b0 means invalid.

struct _mmu_config

#include <fsl_mmu.h> The structure used to initialize the MMU instance includes unit size, page mode, address range, and LUT content.

Public Members

mmu_unit_size_t unitSize
Specify the basic size of unit that is used in the memory map.

mmu_page_option_t pageOpt
Specify the behavior of page option.

mmu_invalid_address_mode_t invalidAddrMode
Specify MMU behavior when an invalid address is detected.

uint16_t baseAddr
Upper 16 bits of valid range's base address.

uint16_t maxAddr
Upper 16 bits of valid range's maximum address.

uint32_t *ptrPhyAddrArray
Pointer to the physical address array used for array mapping.

uint32_t *ptrVirtualAddrArray
Pointer to the virtual address array used for array mapping.

uint16_t countOfAddr
Count of array elements

bool enableMMP
Enable/disable memory map feature, once enabled all AHB access to corresponding address are remapped by LUT SRAM.

2.63 MRT: Multi-Rate Timer

void MRT_Init(MRT_Type *base, const *mrt_config_t* *config)
Ungates the MRT clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the MRT driver.

Parameters

- base – Multi-Rate timer peripheral base address
- config – Pointer to user's MRT config structure. If MRT has MULTITASK bit field in MODCFG reigster, param config is useless.

void MRT_Deinit(MRT_Type *base)
Gate the MRT clock.

Parameters

- base – Multi-Rate timer peripheral base address

static inline void MRT_GetDefaultConfig(*mrt_config_t* *config)

Fill in the MRT config struct with the default settings.

The default values are:

```
config->enableMultiTask = false;
```

Parameters

- config – Pointer to user's MRT config structure.

static inline void MRT_SetupChannelMode(MRT_Type *base, *mrt_chnl_t* channel, const *mrt_timer_mode_t* mode)

Sets up an MRT channel mode.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Channel that is being configured.
- mode – Timer mode to use for the channel.

static inline void MRT_EnableInterrupts(MRT_Type *base, *mrt_chnl_t* channel, uint32_t mask)

Enables the MRT interrupt.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *mrt_interrupt_enable_t*

static inline void MRT_DisableInterrupts(MRT_Type *base, *mrt_chnl_t* channel, uint32_t mask)

Disables the selected MRT interrupt.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number
- mask – The interrupts to disable. This is a logical OR of members of the enumeration *mrt_interrupt_enable_t*

static inline uint32_t MRT_GetEnabledInterrupts(MRT_Type *base, *mrt_chnl_t* channel)

Gets the enabled MRT interrupts.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *mrt_interrupt_enable_t*

static inline uint32_t MRT_GetStatusFlags(MRT_Type *base, *mrt_chnl_t* channel)

Gets the MRT status flags.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number

Returns
The status flags. This is the logical OR of members of the enumeration *mrt_status_flags_t*

static inline void MRT_ClearStatusFlags(MRT_Type *base, *mrt_chnl_t* channel, uint32_t mask)

Clears the MRT status flags.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number
- mask – The status flags to clear. This is a logical OR of members of the enumeration *mrt_status_flags_t*

void MRT_UpdateTimerPeriod(MRT_Type *base, *mrt_chnl_t* channel, uint32_t count, bool immediateLoad)

Used to update the timer period in units of count.

The new value will be immediately loaded or will be loaded at the end of the current time interval. For one-shot interrupt mode the new value will be immediately loaded.

Note: User can call the utility macros provided in fsl_common.h to convert to ticks

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number
- count – Timer period in units of ticks
- immediateLoad – true: Load the new value immediately into the TIMER register; false: Load the new value at the end of current timer interval

static inline uint32_t MRT_GetCurrentTimerCount(MRT_Type *base, *mrt_chnl_t* channel)

Reads the current timer counting value.

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note: User can call the utility macros provided in fsl_common.h to convert ticks to usec or msec

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number

Returns
Current timer counting value in ticks

static inline void MRT_StartTimer(MRT_Type *base, *mrt_chnl_t* channel, uint32_t count)

Starts the timer counting.

After calling this function, timers load period value, counts down to 0 and depending on the timer mode it will either load the respective start value again or stop.

Note: User can call the utility macros provided in fsl_common.h to convert to ticks

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number.
- count – Timer period in units of ticks. Count can contain the LOAD bit, which control the force load feature.

static inline void MRT_StopTimer(MRT_Type *base, *mrt_chnl_t* channel)

Stops the timer counting.

This function stops the timer from counting.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number.

static inline uint32_t MRT_GetIdleChannel(MRT_Type *base)

Find the available channel.

This function returns the lowest available channel number.

Parameters

- base – Multi-Rate timer peripheral base address

static inline void MRT_ReleaseChannel(MRT_Type *base, *mrt_chnl_t* channel)

Release the channel when the timer is using the multi-task mode.

In multi-task mode, the INUSE flags allow more control over when MRT channels are released for further use. The user can hold on to a channel acquired by calling *MRT_GetIdleChannel()* for as long as it is needed and release it by calling this function. This removes the need to ask for an available channel for every use.

Parameters

- base – Multi-Rate timer peripheral base address
- channel – Timer channel number.

FSL_MRT_DRIVER_VERSION

enum _mrt_chnl

List of MRT channels.

Values:

enumerator kMRT_Channel_0

MRT channel number 0

enumerator kMRT_Channel_1

MRT channel number 1

enumerator kMRT_Channel_2

MRT channel number 2

enumerator kMRT_Channel_3

MRT channel number 3

enum _mrt_timer_mode

List of MRT timer modes.

Values:

enumerator kMRT_RepeatMode

Repeat Interrupt mode

enumerator kMRT_OneShotMode

One-shot Interrupt mode

enumerator kMRT_OneShotStallMode

One-shot stall mode

enum _mrt_interrupt_enable

List of MRT interrupts.

Values:

enumerator kMRT_TimerInterruptEnable

Timer interrupt enable

enum _mrt_status_flags

List of MRT status flags.

Values:

enumerator kMRT_TimerInterruptFlag

Timer interrupt flag

enumerator kMRT_TimerRunFlag

Indicates state of the timer

typedef enum *_mrt_chnl* mrt_chnl_t

List of MRT channels.

typedef enum *_mrt_timer_mode* mrt_timer_mode_t

List of MRT timer modes.

typedef enum *_mrt_interrupt_enable* mrt_interrupt_enable_t

List of MRT interrupts.

typedef enum *_mrt_status_flags* mrt_status_flags_t

List of MRT status flags.

typedef struct *_mrt_config* mrt_config_t

MRT configuration structure.

This structure holds the configuration settings for the MRT peripheral. To initialize this structure to reasonable defaults, call the *MRT_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

struct _mrt_config

#include <fsl_mrt.h> MRT configuration structure.

This structure holds the configuration settings for the MRT peripheral. To initialize this structure to reasonable defaults, call the *MRT_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Public Members

bool enableMultiTask

true: Timers run in multi-task mode; false: Timers run in hardware status mode

2.64 MU: Messaging Unit

void MU_Init(MU_Type *base)

Initializes the MU module.

This function enables the MU clock only.

Parameters

- base – MU peripheral base address.

void MU_Deinit(MU_Type *base)

De-initializes the MU module.

This function disables the MU clock only.

Parameters

- base – MU peripheral base address.

static inline void MU_SendMsgNonBlocking(MU_Type *base, uint32_t regIndex, uint32_t msg)

Writes a message to the TX register.

This function writes a message to the specific TX register. It does not check whether the TX register is empty or not. The upper layer should make sure the TX register is empty before calling this function. This function can be used in ISR for better performance.

```
while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { }  Wait for TX0 register empty.
MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG_VAL);  Write message to the TX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – TX register index, see *mu_msg_reg_index_t*.
- msg – Message to send.

status_t MU_SendMsg(MU_Type *base, uint32_t regIndex, uint32_t msg)

Blocks to send a message.

This function waits until the TX register is empty and sends the message. If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and returns kStatus_Timeout.

Parameters

- base – MU peripheral base address.
- regIndex – MU message register, see *mu_msg_reg_index_t*.
- msg – Message to send.

Return values

- kStatus_Success – Message sent successfully.
- kStatus_Timeout – Timeout occurred while waiting for TX register to be empty.

Returns
status_t

static inline uint32_t MU_ReceiveMsgNonBlocking(MU_Type *base, uint32_t regIndex)

Reads a message from the RX register.

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```
uint32_t msg;
while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
{
}  Wait for the RX0 register full.

msg = MU_ReceiveMsgNonBlocking(base, kMU_MsgReg0);  Read message from RX0 register.
```

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see *mu_msg_reg_index_t*.

Returns
The received message.

status_t MU_ReceiveMsgTimeout(MU_Type *base, uint32_t regIndex, uint32_t *readValue)

Blocks to receive a message with timeout protection.

This function waits until the RX register is full and receives the message. If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

This function provides the same blocking behavior as *MU_ReceiveMsg()* but with additional timeout protection to prevent system hangs if the other core becomes unresponsive or if hardware issues occur.

Note: Both *MU_ReceiveMsg()* and *MU_ReceiveMsgTimeout()* are blocking functions. The difference is that this function includes timeout protection while *MU_ReceiveMsg()* waits indefinitely.

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see *mu_msg_reg_index_t*.
- readValue – Pointer to store the received message.

Return values

- kStatus_Success – Message received successfully.

- kStatus_InvalidArgument – Invalid readValue pointer.
- kStatus_Timeout – Timeout occurred while waiting for RX register to be full.

Returns
status_t

uint32_t MU_ReceiveMsg(MU_Type *base, uint32_t regIndex)

Blocks to receive a message (infinite wait, no timeout protection).

This function waits until the RX register is full and receives the message. This function will wait indefinitely until a message is received.

Note: Both *MU_ReceiveMsg()* and *MU_ReceiveMsgTimeout()* are blocking functions. The difference is that *MU_ReceiveMsgTimeout()* includes timeout protection while this function waits indefinitely.

Warning: This function does not include timeout protection and may cause system hangs if the other core becomes unresponsive. For applications requiring timeout protection, use *MU_ReceiveMsgTimeout()* instead.

Parameters

- base – MU peripheral base address.
- regIndex – RX register index, see *mu_msg_reg_index_t*.

Returns
The received message.

static inline void MU_SetFlagsNonBlocking(MU_Type *base, uint32_t flags)

Sets the 3-bit MU flags reflect on the other MU side.

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag kMU_FlagsUpdatingFlag is cleared before calling this function.

```
while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
{
}  Wait for previous MU flags updating.

MU_SetFlagsNonBlocking(base, 0U);  Set the mU flags.
```

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

status_t MU_SetFlags(MU_Type *base, uint32_t flags)

brief Blocks setting the 3-bit MU flags reflect on the other MU side.

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag kMU_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. This

function waits for the MU status flag kMU_FlagsUpdatingFlag cleared and sets the 3-bit MU flags.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

return status_t retval kStatus_Success Flags were set successfully. retval kStatus_Timeout Timeout occurred while waiting for flags to update.

Parameters

- base – MU peripheral base address.
- flags – The 3-bit MU flags to set.

static inline uint32_t MU_GetFlags(MU_Type *base)

Gets the current value of the 3-bit MU flags set by the other side.

This function gets the current 3-bit MU flags on the current side.

Parameters

- base – MU peripheral base address.

Returns

flags Current value of the 3-bit flags.

uint32_t MU_GetStatusFlags(MU_Type *base)

Gets the MU status flags.

This function returns the bit mask of the MU status flags. See _mu_status_flags.

```
uint32_t flags;
flags = MU_GetStatusFlags(base);  Get all status flags.
if (kMU_Tx0EmptyFlag & flags)
{
    The TX0 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG0_VAL);
}
if (kMU_Tx1EmptyFlag & flags)
{
    The TX1 register is empty. Message can be sent.
    MU_SendMsgNonBlocking(base, kMU_MsgReg1, MSG1_VAL);
}
```

If there are more than 4 general purpose interrupts, use *MU_GetGeneralPurposeStatusFlags*.

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU status flags, see _mu_status_flags.

static inline uint32_t MU_GetInterruptsPending(MU_Type *base)

Gets the MU IRQ pending status of enabled interrupts.

This function returns the bit mask of the pending MU IRQs of enabled interrupts. Only these flags are checked. kMU_Tx0EmptyFlag kMU_Tx1EmptyFlag kMU_Tx2EmptyFlag kMU_Tx3EmptyFlag kMU_Rx0FullFlag kMU_Rx1FullFlag kMU_Rx2FullFlag kMU_Rx3FullFlag kMU_GenInt0Flag kMU_GenInt1Flag kMU_GenInt2Flag kMU_GenInt3Flag

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU IRQs pending.

static inline void MU_ClearStatusFlags(MU_Type *base, uint32_t flags)

Clears the specific MU status flags.

This function clears the specific MU status flags. The flags to clear should be passed in as bit mask. See _mu_status_flags.

```
Clear general interrupt 0 and general interrupt 1 pending flags.
MU_ClearStatusFlags(base, kMU_GenInt0Flag | kMU_GenInt1Flag);
```

If there are more than 4 general purpose interrupts, use *MU_ClearGeneralPurposeStatusFlags*.

Parameters

- base – MU peripheral base address.
- flags – Bit mask of the MU status flags. See _mu_status_flags. Only the following flags can be cleared by software (if applicable for particular device), other flags are cleared by hardware:
  - kMU_GenInt0Flag
  - kMU_GenInt1Flag
  - kMU_GenInt2Flag
  - kMU_GenInt3Flag
  - kMU_MuResetInterruptFlag
  - kMU_OtherSideEnterRunInterruptFlag
  - kMU_OtherSideEnterHaltInterruptFlag
  - kMU_OtherSideEnterWaitInterruptFlag
  - kMU_OtherSideEnterStopInterruptFlag
  - kMU_OtherSideEnterPowerDownInterruptFlag
  - kMU_ResetAssertInterruptFlag
  - kMU_HardwareResetInterruptFlag

static inline void MU_EnableInterrupts(MU_Type *base, uint32_t interrupts)

Enables the specific MU interrupts.

This function enables the specific MU interrupts. The interrupts to enable should be passed in as bit mask. See _mu_interrupt_enable.

```
  Enable general interrupt 0 and TX0 empty interrupt.
MU_EnableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

If there are more than 4 general purpose interrupts, use *MU_EnableGeneralPurposeInterrupts*.

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU interrupts. See _mu_interrupt_enable.

static inline void MU_DisableInterrupts(MU_Type *base, uint32_t interrupts)

Disables the specific MU interrupts.

This function disables the specific MU interrupts. The interrupts to disable should be passed in as bit mask. See _mu_interrupt_enable.

```
  Disable general interrupt 0 and TX0 empty interrupt.
MU_DisableInterrupts(base, kMU_GenInt0InterruptEnable | kMU_Tx0EmptyInterruptEnable);
```

If there are more than 4 general purpose interrupts, use *MU_DisableGeneralPurposeInterrupts*.

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU interrupts. See _mu_interrupt_enable.

status_t MU_TriggerInterrupts(MU_Type *base, uint32_t interrupts)

Triggers interrupts to the other core.

This function triggers the specific interrupts to the other core. The interrupts to trigger are passed in as bit mask. See *_mu_interrupt_trigger*. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
if (kStatus_Success != MU_TriggerInterrupts(base, kMU_GenInt0InterruptTrigger | kMU_
↪GenInt2InterruptTrigger))
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

If there are more than 4 general purpose interrupts, use *MU_TriggerGeneralPurposeInterrupts*.

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the interrupts to trigger. See _mu_interrupt_trigger.

Return values

- kStatus_Success – Interrupts have been triggered successfully.
- kStatus_Fail – Previous interrupts have not been accepted.

static inline void MU_EnableGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Enables the MU general purpose interrupts.

This function enables the MU general purpose interrupts. The interrupts to enable should be passed in as bit mask of *mu_general_purpose_interrupt_t*. The function *MU_EnableInterrupts* only support general interrupt 0~3, this function supports all general interrupts.

For example, to enable general purpose interrupt 0 and 3, use like this:

```
MU_EnableGeneralPurposeInterrupts(MU, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt3);
```

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU general purpose interrupts, see *mu_general_purpose_interrupt_t*.

static inline void MU_DisableGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Disables the MU general purpose interrupts.

This function disables the MU general purpose interrupts. The interrupts to disable should be passed in as bit mask of *mu_general_purpose_interrupt_t*. The function *MU_DisableInterrupts* only support general interrupt 0~3, this function supports all general interrupts.

For example, to disable general purpose interrupt 0 and 3, use like this:

```
MU_EnableGeneralPurposeInterrupts(MU, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt3);
```

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the MU general purpose interrupts. see *mu_general_purpose_interrupt_t*.

static inline uint32_t MU_GetGeneralPurposeStatusFlags(MU_Type *base)

Gets the MU general purpose interrupt status flags.

This function returns the bit mask of the MU general purpose interrupt status flags. *MU_GetStatusFlags* can only get general purpose interrupt status 0~3, this function can get all general purpose interrupts status.

This example shows to check whether general purpose interrupt 0 and 3 happened.

```
uint32_t flags;
flags = MU_GetGeneralPurposeStatusFlags(base);
if (kMU_GeneralPurposeInterrupt0 & flags)
{
}
if (kMU_GeneralPurposeInterrupt3 & flags)
{
}
```

Parameters

- base – MU peripheral base address.

Returns

Bit mask of the MU general purpose interrupt status flags.

static inline void MU_ClearGeneralPurposeStatusFlags(MU_Type *base, uint32_t flags)

Clear the MU general purpose interrupt status flags.

This function clears the specific MU general purpose interrupt status flags. The flags to clear should be passed in as bit mask. mu_general_purpose_interrupt_t_mu_status_flags.

Example to clear general purpose interrupt 0 and general interrupt 1 pending flags.

```
MU_ClearGeneralPurposeStatusFlags(base, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt1);
```

Parameters

- base – MU peripheral base address.
- flags – Bit mask of the MU general purpose interrupt status flags. See *mu_general_purpose_interrupt_t*.

static inline uint32_t MU_GetRxStatusFlags(MU_Type *base)

Return the RX status flags in reverse numerical order.

This function return the RX status flags in reverse order. Note: RFn bits of SR[3-0](mu status register) are mapped in ascending numerical order: RF0 -> SR[0] RF1 -> SR[1] RF2 -> SR[2] RF3 -> SR[3] This function will return these bits in reverse numerical order(RF3->RF1) to comply with *MU_GetRxStatusFlags()* of mu driver. See *MU_GetRxStatusFlags()* from drivers/mu/fsl_mu.h

```
status_reg = MU_GetRxStatusFlags(base);
```

Parameters

- base – MU peripheral base address.

Returns

MU RX status flags in reverse order

status_t MU_TriggerGeneralPurposeInterrupts(MU_Type *base, uint32_t interrupts)

Triggers general purpose interrupts to the other core.

This function triggers the specific general purpose interrupts to the other core. The interrupts to trigger are passed in as bit mask. See *mu_general_purpose_interrupt_t*. The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
status_t status;
status = MU_TriggerGeneralPurposeInterrupts(base, kMU_GeneralPurposeInterrupt0 | kMU_
↪GeneralPurposeInterrupt2);

if (kStatus_Success != status)
{
    Previous general purpose interrupt 0 or general purpose interrupt 2
    has not been processed by the other core.
}
```

Parameters

- base – MU peripheral base address.
- interrupts – Bit mask of the interrupts to trigger. See *mu_general_purpose_interrupt_t*.

Return values

- kStatus_Success – Interrupts have been triggered successfully.
- kStatus_Fail – Previous interrupts have not been accepted.

void MU_BootOtherCore(MU_Type *base, mu_core_boot_mode_t mode)

Boots the other core.

This function boots the other core with a boot configuration.

Parameters

- base – MU peripheral base address.
- mode – The other core boot mode.

void MU_HoldOtherCoreReset(MU_Type *base)

Holds the other core reset.

This function causes the other core to be held in reset following any reset event.

Parameters

- base – MU peripheral base address.

static inline *status_t* MU_ResetBothSides(MU_Type *base)

Resets the MU for both A side and B side.

This function resets the MU for both A side and B side. Before reset, it is recommended to interrupt processor B, because this function may affect the ongoing processor B programs.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations if waiting for the other side to come out of reset takes too long.

Note: For some platforms, only MU side A could use this function, check reference manual for details.

Parameters

- base – MU peripheral base address.

Return values

- kStatus_Success – The MU was reset successfully.
- kStatus_Timeout – Timeout occurred while waiting for the other side to come out of reset.

Returns

status_t

status_t MU_HardwareResetOtherCore(MU_Type *base, bool waitReset, bool holdReset, mu_core_boot_mode_t bootMode)

Hardware reset the other core.

This function resets the other core, the other core could mask the hardware reset by calling MU_MaskHardwareReset. The hardware reset mask feature is only available for some platforms. This function could be used together with MU_BootOtherCore to control the other core reset workflow.

If MU1_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout if waiting for the other core to enter or exit reset takes too long.

Example 1: Reset the other core, and no hold reset

```
MU_HardwareResetOtherCore(MU_A, true, false, bootMode);
```

In this example, the core at MU side B will reset with the specified boot mode.

Example 2: Reset the other core and hold it, then boot the other core later. Here the other core enters reset, and the reset is hold

```
MU_HardwareResetOtherCore(MU_A, true, true, modeDontCare);
```

Current core boot the other core when necessary.

```
MU_BootOtherCore(MU_A, bootMode);
```

Note: The feature waitReset, holdReset, and bootMode might be not supported for some platforms. waitReset is only available for platforms that FSL_FEATURE_MU_NO_CORE_STATUS not defined as 1 and

FSL_FEATURE_MU_HAS_RESET_ASSERT_INT not defined as 0. holdReset is only available for platforms that FSL_FEATURE_MU_HAS_RSTH not defined as 0. bootMode is only available for platforms that FSL_FEATURE_MU_HAS_BOOT not defined as 0.

Parameters

- base – MU peripheral base address.
- waitReset – Wait the other core enters reset. Only work when there is CSSR0[RAIP].
  - true: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked.
  - false: Don't wait the reset.
- holdReset – Hold the other core reset or not. Only work when there is CCR0[RSTH].
  - true: Hold the other core in reset, this function returns directly when the other core enters reset.
  - false: Don't hold the other core in reset, this function waits until the other core out of reset.
- bootMode – Boot mode of the other core, if holdReset is true, this parameter is useless.

Return values

- kStatus_Success – The other core was reset successfully.
- kStatus_Timeout – Timeout occurred while waiting for the other core to enter or exit reset.

Returns

status_t

FSL_MU_DRIVER_VERSION

MU driver version.

enum _mu_status_flags

MU status flags.

Values:

enumerator kMU_Tx0EmptyFlag

TX0 empty.

enumerator kMU_Tx1EmptyFlag

TX1 empty.

enumerator kMU_Tx2EmptyFlag

TX2 empty.

enumerator kMU_Tx3EmptyFlag

TX3 empty.

enumerator kMU_Rx0FullFlag
RX0 full.

enumerator kMU_Rx1FullFlag
RX1 full.

enumerator kMU_Rx2FullFlag
RX2 full.

enumerator kMU_Rx3FullFlag
RX3 full.

enumerator kMU_GenInt0Flag
General purpose interrupt 0 pending.

enumerator kMU_GenInt1Flag
General purpose interrupt 1 pending.

enumerator kMU_GenInt2Flag
General purpose interrupt 2 pending.

enumerator kMU_GenInt3Flag
General purpose interrupt 3 pending.

enumerator kMU_RxFullPendingFlag
Any RX full flag is pending.

enumerator kMU_TxEmptyPendingFlag
Any TX empty flag is pending.

enumerator kMU_GenIntPendingFlag
Any general interrupt flag is pending.

enumerator kMU_EventPendingFlag
MU event pending.

enumerator kMU_FlagsUpdatingFlag
MU flags update is on-going.

enumerator kMU_MuInResetFlag
MU of any side is in reset.

enumerator kMU_MuResetInterruptFlag
The other side initializes MU reset.

enum __mu_interrupt_enable
MU interrupt source to enable.
Values:

enumerator kMU_Tx0EmptyInterruptEnable
TX0 empty.

enumerator kMU_Tx1EmptyInterruptEnable
TX1 empty.

enumerator kMU_Tx2EmptyInterruptEnable
TX2 empty.

enumerator kMU_Tx3EmptyInterruptEnable
TX3 empty.

enumerator kMU_Rx0FullInterruptEnable
RX0 full.

enumerator kMU_Rx1FullInterruptEnable
RX1 full.

enumerator kMU_Rx2FullInterruptEnable
RX2 full.

enumerator kMU_Rx3FullInterruptEnable
RX3 full.

enumerator kMU_GenInt0InterruptEnable
General purpose interrupt 0.

enumerator kMU_GenInt1InterruptEnable
General purpose interrupt 1.

enumerator kMU_GenInt2InterruptEnable
General purpose interrupt 2.

enumerator kMU_GenInt3InterruptEnable
General purpose interrupt 3.

enumerator kMU_MuResetInterruptEnable
The other side initializes MU reset.

enum _mu_interrupt_trigger
MU interrupt that could be triggered to the other core.
Values:

enumerator kMU_GenInt0InterruptTrigger
General purpose interrupt 0.

enumerator kMU_GenInt1InterruptTrigger
General purpose interrupt 1.

enumerator kMU_GenInt2InterruptTrigger
General purpose interrupt 2.

enumerator kMU_GenInt3InterruptTrigger
General purpose interrupt 3.

enum _mu_msg_reg_index
MU message register index.
Values:

enumerator kMU_MsgReg0
Message register 0.

enumerator kMU_MsgReg1
Message register 1.

enumerator kMU_MsgReg2
Message register 2.

enumerator kMU_MsgReg3
Message register 3.

enum _mu_general_purpose_interrupt
MU general purpose interrupts.
Values:

enumerator kMU_GeneralPurposeInterrupt0
General purpose interrupt 0

enumerator kMU_GeneralPurposeInterrupt1
General purpose interrupt 1

enumerator kMU_GeneralPurposeInterrupt2
General purpose interrupt 2

enumerator kMU_GeneralPurposeInterrupt3
General purpose interrupt 3

typedef enum *_mu_msg_reg_index* mu_msg_reg_index_t
MU message register index.

typedef enum *_mu_general_purpose_interrupt* mu_general_purpose_interrupt_t
MU general purpose interrupts.

MU_CORE_INTR(intr)

MU_MISC_INTR(intr)

MU_TX_INTR(intr)

MU_RX_INTR(intr)

MU_GI_INTR(intr)

MU_GET_CORE_INTR(intrs)

MU_GET_TX_INTR(intrs)

MU_GET_RX_INTR(intrs)

MU_GET_GI_INTR(intrs)

MU_CORE_FLAG(flag)

MU_STAT_FLAG(flag)

MU_TX_FLAG(flag)

MU_RX_FLAG(flag)

MU_GI_FLAG(flag)

MU_GET_CORE_FLAG(flags)

MU_GET_STAT_FLAG(flags)

MU_GET_TX_FLAG(flags)

MU_GET_RX_FLAG(flags)

MU_GET_GI_FLAG(flags)

MU1_BUSY_POLL_COUNT

Maximum polling iterations for MU waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the MU code before timing out and returning an error.

It applies to all waiting loops in MU driver, such as waiting for TX register to be empty or waiting for RX register to be full.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if a core becomes unresponsive.

2.65 This type defines status return values used by NBOOT functions that are not easily

kStatus_NBOOT_Success

Operation completed successfully.

kStatus_NBOOT_Fail

Operation failed.

kStatus_NBOOT_InvalidArgument

Invalid argument passed to the function.

kStatus_NBOOT_RequestTimeout
: Operation timed out.

kStatus_NBOOT_KeyNotLoaded
: The requested key is not loaded.

kStatus_NBOOT_AuthFail
: Authentication failed.

kStatus_NBOOT_OperationNotAvaialable
: Operation not available on this HW.

kStatus_NBOOT_KeyNotAvailable
: Key is not avaialble.

kStatus_NBOOT_IvCounterOverflow
: Overflow of IV counter (PRINCE/IPED).

kStatus_NBOOT_SelftestFail
: FIPS self-test failure.

kStatus_NBOOT_InvalidDataFormat
: Invalid data format for example antipole

kStatus_NBOOT_IskCertUserDataTooBig
: Size of User data in ISK certificate is greater than 96 bytes

kStatus_NBOOT_IskCertSignatureOffsetTooSmall
: Signature offset in ISK certificate is smaller than expected

kStatus_NBOOT_MemcpyFail
: Unexpected error detected during nboot_memcpy()

2.66 OSTIMER: OS Event Timer Driver

void OSTIMER_Init(OSTIMER_Type *base)
: Initializes an OSTIMER by turning its bus clock on.

void OSTIMER_Deinit(OSTIMER_Type *base)
: Deinitializes a OSTIMER instance.

  This function shuts down OSTIMER bus clock

  Parameters

  - base – OSTIMER peripheral base address.

uint64_t OSTIMER_GrayToDecimal(uint64_t gray)
: Translate the value from gray-code to decimal.

  Parameters

  - gray – The gray value input.

Returns
The decimal value.

static inline uint64_t OSTIMER_DecimalToGray(uint64_t dec)

Translate the value from decimal to gray-code.

Parameters

- dec – The decimal value.

Returns
The gray code of the input value.

uint32_t OSTIMER_GetStatusFlags(OSTIMER_Type *base)

Get OSTIMER status Flags.

This returns the status flag. Currently, only match interrupt flag can be got.

Parameters

- base – OSTIMER peripheral base address.

Returns
status register value

void OSTIMER_ClearStatusFlags(OSTIMER_Type *base, uint32_t mask)

Clear Status Interrupt Flags.

This clears interrupt status flag. Currently, only match interrupt flag can be cleared.

Parameters

- base – OSTIMER peripheral base address.
- mask – Clear bit mask.

Returns
none

status_t OSTIMER_SetMatchRawValue(OSTIMER_Type *base, uint64_t count, *ostimer_callback_t* cb)

Set the match raw value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central EVTIMER. Please note that, the data format may be gray-code, if so, please use *OSTIMER_SetMatchValue()*.

Parameters

- base – OSTIMER peripheral base address.
- count – OSTIMER timer match value.(Value may be gray-code format)
- cb – OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values
kStatus_Success – Match raw value written and interrupt enabled successfully.

status_t OSTIMER_SetMatchValue(OSTIMER_Type *base, uint64_t count, *ostimer_callback_t* cb)

Set the match value for OSTIMER.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central EVTIMER.

The function disables the match interrupt before writing the match registers and re-enables it immediately after. This function is suitable when the delta between the

current timer value and count is comfortably larger than the match-register synchronisation latency (more than 7 OSTimer ticks). However, it does NOT wait for OSEVENT_CTRL[MATCH_WR_RDY]) after writing MATCH, and it does NOT check whether the requested match time has already passed. If there is any possibility that count is close to or has already passed the current timer value, use *OSTIMER_SetMatchValueSafe()* instead, which polls MATCH_WR_RDY and checks whether the match moment has been missed.

Parameters

- base – OSTIMER peripheral base address.
- count – Match value in decimal (binary) format. The driver converts to Gray code internally when the hardware counter is Gray-encoded.
- cb – OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

kStatus_Success – Match value written and interrupt enabled successfully.

status_t OSTIMER_SetMatchValueSafe(OSTIMER_Type *base, uint64_t count, *ostimer_callback_t* cb)

Set the match value for OSTIMER with full synchronisation and missed-event detection.

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central EVTIMER.

Unlike *OSTIMER_SetMatchValue()*, this function performs the following additional steps after writing the match registers:

a. Wait for write synchronisation (MATCH_WR_RDY). On devices that expose OSEVENT_CTRL[MATCH_WR_RDY], the function spins until that bit is cleared, which indicates that the written value has been transferred from the shadow registers to the active compare registers in the OSTimer clock domain. This eliminates the race condition where the timer advances past the match value before the hardware has latched it.

b. Check whether the match moment has already been missed. After MATCH_WR_RDY clears, the function reads the current timer value and compares it with count:

   - If the current timer value is still below count, the match has not yet occurred. The interrupt is enabled and the function returns *kStatus_Success*.
   - If the current timer value has reached or passed count:
     - If the hardware interrupt flag (OSTIMER_INTRFLAG) is already set, the match event was captured by hardware. The interrupt is enabled so the pending flag triggers the ISR, and the function returns *kStatus_Success*.
     - If the interrupt flag is NOT set, the match moment passed without the hardware capturing it (the compare logic had not yet latched the value when the timer advanced). The interrupt is left DISABLED and the function returns *kStatus_Fail*. The caller is responsible for handling this missed event, for example by scheduling a new match immediately or executing the intended action directly.

Note: This function is recommended whenever the requested match delta is small or unpredictable relative to the OSTimer source clock period. If the overhead of polling MATCH_WR_RDY is unacceptable (e.g. at 32 KHz with tight real-time constraints), use *OSTIMER_SetMatchValue()* and ensure the match delta is always large enough.

Parameters

- base – OSTIMER peripheral base address.
- count – Match value in decimal (binary) format. The driver converts to Gray code internally when the hardware counter is Gray-encoded.
- cb – OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

- kStatus_Success – Match value written and interrupt enabled. The match event will occur in the future, or the hardware interrupt flag was already set and the pending interrupt will fire immediately upon enabling.
- kStatus_Fail – Match time already passed and the hardware interrupt flag was not set. The interrupt remains disabled. The caller must handle the missed event.

static inline void OSTIMER_SetMatchRegister(OSTIMER_Type *base, uint64_t value)

Set value to OSTIMER MATCH register directly.

This function writes the input value to OSTIMER MATCH register directly, it does not touch any other registers. Note that, the data format is gray-code if the hardware register is gray-encoded. The function *OSTIMER_DecimalToGray* could convert decimal value to gray code.

Parameters

- base – OSTIMER peripheral base address.
- value – OSTIMER timer match value (Value is gray-code format if the hardware register is gray-encoded).

static inline uint64_t OSTIMER_GetMatchRegister(OSTIMER_Type *base)

Get the match value from OSTIMER.

This function will get the match value from OSTIMER. The value of timer match is gray code format if the hardware is Gray-encoded.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of match register, data format is gray code if the register is Gray-encoded.

static inline uint64_t OSTIMER_GetMatchValue(OSTIMER_Type *base)

Get the match value from OSTIMER.

This function will get a match value from OSTIMER.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of match register in decimal format.

static inline void OSTIMER_EnableMatchInterrupt(OSTIMER_Type *base)

Enable the OSTIMER counter match interrupt.

Enable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

- base – OSTIMER peripheral base address.

static inline void OSTIMER_DisableMatchInterrupt(OSTIMER_Type *base)

Disable the OSTIMER counter match interrupt.

Disable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

- base – OSTIMER peripheral base address.

static inline uint64_t OSTIMER_GetCurrentTimerRawValue(OSTIMER_Type *base)

Get current timer raw count value from OSTIMER.

This function will get the timer count value from OS timer register. The raw value of timer count may be gray code format.

Parameters

- base – OSTIMER peripheral base address.

Returns

Raw value of OSTIMER, may be gray code format if the hardware register is Gray-encoded.

uint64_t OSTIMER_GetCurrentTimerValue(OSTIMER_Type *base)

Get current timer count value from OSTIMER.

This function will get a decimal timer count value. If the RAW value of timer count is gray code format, it will be translated to decimal data internally.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of OSTIMER which will be formated to decimal value.

static inline uint64_t OSTIMER_GetCaptureRawValue(OSTIMER_Type *base)

Get the capture value from OSTIMER.

This function will get a captured value from OSTIMER. The Raw value of timer capture may be gray code format if the hardware register is Gray-encoded.

Parameters

- base – OSTIMER peripheral base address.

Returns

Raw value of capture register, data format may be gray code if the hardware register is Gray-encoded.

uint64_t OSTIMER_GetCaptureValue(OSTIMER_Type *base)

Get the capture value from OSTIMER.

This function will get a capture decimal-value from OSTIMER. If the RAW value of timer count is gray code format, it will be translated to decimal data internally.

Parameters

- base – OSTIMER peripheral base address.

Returns

Value of capture register, data format is decimal.

void OSTIMER_HandleIRQ(OSTIMER_Type *base, *ostimer_callback_t* cb)

OS timer interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in *OSTIMER_SetMatchValue()*). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

- base – OS timer peripheral base address.
- cb – callback scheduled for this instance of OS timer

Returns
none

void OSTIMER_DriverIRQHandler(uint32_t instance)

OSTIMER parameterized IRQ handler entry.

Parameters

- instance – OSTIMER instance index.

FSL_OSTIMER_DRIVER_VERSION

OSTIMER driver version.

enum _ostimer_flags

OSTIMER status flags.

Values:

enumerator kOSTIMER_MatchInterruptFlag

Match interrupt flag bit, sets if the match value was reached.

typedef void (*ostimer_callback_t)(void)

ostimer callback function.

2.67 PDM: Microphone Interface

2.68 PDM Driver

void PDM_Init(PDM_Type *base, const *pdm_config_t* *config)

Initializes the PDM peripheral.

Ungates the PDM clock, resets the module, and configures PDM with a configuration structure. The configuration structure can be custom filled or set with default values by PDM_GetDefaultConfig().

Note: This API should be called at the beginning of the application to use the PDM driver. Otherwise, accessing the PDM module can cause a hard fault because the clock is not enabled.

Parameters

- base – PDM base pointer
- config – PDM configuration structure.

void PDM_Deinit(PDM_Type *base)

De-initializes the PDM peripheral.

This API gates the PDM clock. The PDM module can't operate unless PDM_Init is called to enable the clock.

Parameters

- base – PDM base pointer

static inline void PDM_Reset(PDM_Type *base)

Resets the PDM module.

Parameters

- base – PDM base pointer

static inline void PDM_Enable(PDM_Type *base, bool enable)

Enables/disables PDM interface.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is enabled, false means PDM interface is disabled.

static inline void PDM_EnableDebugMode(PDM_Type *base, bool enable)

Enables/disables debug mode for PDM. The PDM interface cannot enter debug mode once in Disable/Low Leakage or Low Power mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface enter debug mode, false means PDM interface in normal mode.

static inline void PDM_EnableInDebugMode(PDM_Type *base, bool enable)

Enables/disables PDM interface in debug mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is enabled debug mode, false means PDM interface is disabled after after completing the current frame in debug mode.

static inline void PDM_EnterLowLeakageMode(PDM_Type *base, bool enable)

Enables/disables PDM interface disable/Low Leakage mode.

Parameters

- base – PDM base pointer
- enable – True means PDM interface is in disable/low leakage mode, False means PDM interface is in normal mode.

static inline void PDM_EnableChannel(PDM_Type *base, uint8_t channel, bool enable)

Enables/disables the PDM channel.

Parameters

- base – PDM base pointer
- channel – PDM channel number need to enable or disable.
- enable – True means enable PDM channel, false means disable.

void PDM_SetChannelConfig(PDM_Type *base, uint32_t channel, const *pdm_channel_config_t* *config)

PDM one channel configurations.

Parameters

- base – PDM base pointer
- config – PDM channel configurations.
- channel – channel number. after completing the current frame in debug mode.

status_t PDM_SetSampleRateConfig(PDM_Type *base, uint32_t sourceClock_HZ, uint32_t sampleRate_HZ)

PDM set sample rate.

Note: This function is depend on the configuration of the PDM and PDM channel, so the correct call sequence is

```
PDM_Init(base, pdmConfig)
PDM_SetChannelConfig(base, channel, &channelConfig)
PDM_SetSampleRateConfig(base, source, sampleRate)
```

Parameters

- base – PDM base pointer
- sourceClock_HZ – PDM source clock frequency.
- sampleRate_HZ – PDM sample rate.

status_t PDM_SetSampleRate(PDM_Type *base, uint32_t enableChannelMask, *pdm_df_quality_mode_t* qualityMode, uint8_t osr, uint32_t clkDiv)

PDM set sample rate.

Deprecated:

Do not use this function. It has been superceded by *PDM_SetSampleRateConfig*

Parameters

- base – PDM base pointer
- enableChannelMask – PDM channel enable mask.
- qualityMode – quality mode.
- osr – cic oversample rate
- clkDiv – clock divider

uint32_t PDM_GetInstance(PDM_Type *base)

Get the instance number for PDM.

Parameters

- base – PDM base pointer.

static inline uint32_t PDM_GetStatus(PDM_Type *base)

Gets the PDM internal status flag. Use the Status Mask in _pdm_internal_status to get the status value needed.

Parameters

- base – PDM base pointer

Returns

PDM status flag value.

static inline uint32_t PDM_GetFifoStatus(PDM_Type *base)

Gets the PDM FIFO status flag. Use the Status Mask in _pdm_fifo_status to get the status value needed.

Parameters

- base – PDM base pointer

Returns

FIFO status.

static inline uint32_t PDM_GetRangeStatus(PDM_Type *base)

Gets the PDM Range status flag. Use the Status Mask in _pdm_range_status to get the status value needed.

Parameters

- base – PDM base pointer

Returns

output status.

static inline void PDM_ClearStatus(PDM_Type *base, uint32_t mask)

Clears the PDM Tx status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status between kPDM_StatusFrequencyLow and kPDM_StatusCh7FifoDataAvaliable.

static inline void PDM_ClearFIFOStatus(PDM_Type *base, uint32_t mask)

Clears the PDM Tx status.

Parameters

- base – PDM base pointer
- mask – State mask.It can be a combination of the status in _pdm_fifo_status.

static inline void PDM_ClearRangeStatus(PDM_Type *base, uint32_t mask)

Clears the PDM range status.

Parameters

- base – PDM base pointer
- mask – State mask. It can be a combination of the status in _pdm_range_status.

void PDM_EnableInterrupts(PDM_Type *base, uint32_t mask)

Enables the PDM interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM_ErrorInterruptEnable
  - kPDM_FIFOInterruptEnable

static inline void PDM_DisableInterrupts(PDM_Type *base, uint32_t mask)

Disables the PDM interrupt requests.

Parameters

- base – PDM base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kPDM_ErrorInterruptEnable
  - kPDM_FIFOInterruptEnable

static inline void PDM_EnableDMA(PDM_Type *base, bool enable)

Enables/disables the PDM DMA requests.

Parameters

- base – PDM base pointer
- enable – True means enable DMA, false means disable DMA.

static inline uint32_t PDM_GetDataRegisterAddress(PDM_Type *base, uint32_t channel)

Gets the PDM data register address.

This API is used to provide a transfer address for the PDM DMA transfer configuration.

Parameters

- base – PDM base pointer.
- channel – Which data channel used.

Returns

data register address.

void PDM_ReadFifo(PDM_Type *base, uint32_t startChannel, uint32_t channelNums, void *buffer, size_t size, uint32_t dataWidth)

PDM read fifo.

Note: : This function support 16 bit only for IP version that only supports 16bit.

Parameters

- base – PDM base pointer.
- startChannel – start channel number.
- channelNums – total enabled channelnums.
- buffer – received buffer address.
- size – number of samples to read.
- dataWidth – sample width.

void PDM_SetChannelGain(PDM_Type *base, uint32_t channel, *pdm_df_output_gain_t* gain)

Set the PDM channel gain.

Please note for different quality mode, the valid gain value is different, reference RM for detail.

Parameters

- base – PDM base pointer.
- channel – PDM channel index.
- gain – channel gain, the register gain value range is 0 - 15.

void PDM_TransferCreateHandle(PDM_Type *base, *pdm_handle_t* *handle, *pdm_transfer_callback_t* callback, void *userData)

Initializes the PDM handle.

This function initializes the handle for the PDM transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – PDM base pointer.
- handle – PDM handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

status_t PDM_TransferSetChannelConfig(PDM_Type *base, *pdm_handle_t* *handle, uint32_t channel, const *pdm_channel_config_t* *config, uint32_t format)

PDM set channel transfer config.

Parameters

- base – PDM base pointer.
- handle – PDM handle pointer.
- channel – PDM channel.
- config – channel config.
- format – data format, support data width configurations,_pdm_data_width.

Return values

kStatus_PDM_ChannelConfig_Failed – or kStatus_Success.

status_t PDM_TransferReceiveNonBlocking(PDM_Type *base, *pdm_handle_t* *handle, *pdm_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on PDM.

Note: This API returns immediately after the transfer initiates. Call the PDM_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_PDM_Busy, the transfer is finished.

Parameters

- base – PDM base pointer
- handle – Pointer to the pdm_handle_t structure which stores the transfer state.
- xfer – Pointer to the pdm_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_PDM_Busy – Previous receive still not finished.

void PDM_TransferAbortReceive(PDM_Type *base, *pdm_handle_t* *handle)

Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – PDM base pointer
- handle – Pointer to the pdm_handle_t structure which stores the transfer state.

void PDM_TransferHandleIRQ(PDM_Type *base, *pdm_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – PDM base pointer.
- handle – Pointer to the pdm_handle_t structure.

FSL_PDM_DRIVER_VERSION

Version 2.9.6

PDM return status.

Values:

enumerator kStatus_PDM_Busy

PDM is busy.

enumerator kStatus_PDM_CLK_LOW

PDM clock frequency low

enumerator kStatus_PDM_FIFO_ERROR

PDM FIFO underrun or overflow

enumerator kStatus_PDM_QueueFull

PDM FIFO underrun or overflow

enumerator kStatus_PDM_Idle

PDM is idle

enumerator kStatus_PDM_Output_ERROR

PDM is output error

enumerator kStatus_PDM_ChannelConfig_Failed

PDM channel config failed

enum _pdm_interrupt_enable

The PDM interrupt enable flag.

Values:

enumerator kPDM_ErrorInterruptEnable

PDM channel error interrupt enable.

enumerator kPDM_FIFOInterruptEnable

PDM channel FIFO interrupt

enum _pdm_internal_status

The PDM status.

Values:

enumerator kPDM_StatusDfBusyFlag

Decimation filter is busy processing data

enumerator kPDM_StatusFrequencyLow

Mic app clock frequency not high enough

enumerator kPDM_StatusCh0FifoDataAvaliable

channel 0 fifo data reached watermark level

enumerator kPDM_StatusCh1FifoDataAvaliable

channel 1 fifo data reached watermark level

enumerator kPDM_StatusCh2FifoDataAvaliable

channel 2 fifo data reached watermark level

enumerator kPDM_StatusCh3FifoDataAvaliable

channel 3 fifo data reached watermark level

enum _pdm_channel_enable_mask

PDM channel enable mask.

Values:

enumerator kPDM_EnableChannel0

channgel 0 enable mask

enumerator kPDM_EnableChannel1

channgel 1 enable mask

enumerator kPDM_EnableChannel2

channgel 2 enable mask

enumerator kPDM_EnableChannel3

channgel 3 enable mask

enumerator kPDM_EnableChannelAll

enum _pdm_fifo_status

The PDM fifo status.

Values:

enumerator kPDM_FifoStatusUnderflowCh0

channel0 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh1
channel1 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh2
channel2 fifo status underflow

enumerator kPDM_FifoStatusUnderflowCh3
channel3 fifo status underflow

enumerator kPDM_FifoStatusOverflowCh0
channel0 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh1
channel1 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh2
channel2 fifo status overflow

enumerator kPDM_FifoStatusOverflowCh3
channel3 fifo status overflow

enum _pdm_range_status
The PDM output status.
Values:

enumerator kPDM_RangeStatusUnderFlowCh0
channel0 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh1
channel1 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh2
channel2 range status underflow

enumerator kPDM_RangeStatusUnderFlowCh3
channel3 range status underflow

enumerator kPDM_RangeStatusOverFlowCh0
channel0 range status overflow

enumerator kPDM_RangeStatusOverFlowCh1
channel1 range status overflow

enumerator kPDM_RangeStatusOverFlowCh2
channel2 range status overflow

enumerator kPDM_RangeStatusOverFlowCh3
channel3 range status overflow

enum _pdm_dc_remover

PDM DC remover configurations.

Values:

enumerator kPDM_DcRemoverCutOff20Hz

DC remover cut off 20HZ

enumerator kPDM_DcRemoverCutOff13Hz

DC remover cut off 13.3HZ

enumerator kPDM_DcRemoverCutOff40Hz

DC remover cut off 40HZ

enumerator kPDM_DcRemoverBypass

DC remover bypass

enum _pdm_df_quality_mode

PDM decimation filter quality mode.

Values:

enumerator kPDM_QualityModeMedium

quality mode memdium

enumerator kPDM_QualityModeHigh

quality mode high

enumerator kPDM_QualityModeLow

quality mode low

enumerator kPDM_QualityModeVeryLow0

quality mode very low0

enumerator kPDM_QualityModeVeryLow1

quality mode very low1

enumerator kPDM_QualityModeVeryLow2

quality mode very low2

enum _pdm_qulaity_mode_k_factor

PDM quality mode K factor.

Values:

enumerator kPDM_QualityModeHighKFactor

high quality mode K factor = 1 / 2

enumerator kPDM_QualityModeMediumKFactor

medium/very low0 quality mode K factor = 2 / 2

enumerator kPDM_QualityModeLowKFactor
low/very low1 quality mode K factor = 4 / 2

enumerator kPDM_QualityModeVeryLow2KFactor
very low2 quality mode K factor = 8 / 2

enum _pdm_df_output_gain
PDM decimation filter output gain.
Values:

enumerator kPDM_DfOutputGain0
Decimation filter output gain 0

enumerator kPDM_DfOutputGain1
Decimation filter output gain 1

enumerator kPDM_DfOutputGain2
Decimation filter output gain 2

enumerator kPDM_DfOutputGain3
Decimation filter output gain 3

enumerator kPDM_DfOutputGain4
Decimation filter output gain 4

enumerator kPDM_DfOutputGain5
Decimation filter output gain 5

enumerator kPDM_DfOutputGain6
Decimation filter output gain 6

enumerator kPDM_DfOutputGain7
Decimation filter output gain 7

enumerator kPDM_DfOutputGain8
Decimation filter output gain 8

enumerator kPDM_DfOutputGain9
Decimation filter output gain 9

enumerator kPDM_DfOutputGain10
Decimation filter output gain 10

enumerator kPDM_DfOutputGain11
Decimation filter output gain 11

enumerator kPDM_DfOutputGain12
Decimation filter output gain 12

enumerator kPDM_DfOutputGain13
Decimation filter output gain 13

enumerator kPDM_DfOutputGain14
Decimation filter output gain 14

enumerator kPDM_DfOutputGain15
Decimation filter output gain 15

enum _pdm_data_width
PDM data width.
Values:

enumerator kPDM_DataWwidth24
PDM data width 24bit

enumerator kPDM_DataWwidth32
PDM data width 32bit

typedef enum *_pdm_dc_remover* pdm_dc_remover_t
PDM DC remover configurations.

typedef enum *_pdm_df_quality_mode* pdm_df_quality_mode_t
PDM decimation filter quality mode.

typedef enum *_pdm_df_output_gain* pdm_df_output_gain_t
PDM decimation filter output gain.

typedef struct *_pdm_channel_config* pdm_channel_config_t
PDM channel configurations.

typedef struct *_pdm_config* pdm_config_t
PDM user configuration structure.

typedef struct *_pdm_transfer* pdm_transfer_t
PDM SDMA transfer structure.

typedef struct *_pdm_handle* pdm_handle_t
PDM handle.

typedef void (*pdm_transfer_callback_t)(PDM_Type *base, *pdm_handle_t* *handle, *status_t* status, void *userData)
PDM transfer callback prototype.

PDM_XFER_QUEUE_SIZE
PDM XFER QUEUE SIZE.

struct _pdm_channel_config
#include <fsl_pdm.h> PDM channel configurations.

Public Members

pdm_dc_remover_t outputCutOffFreq
PDM output DC remover cut off frequency

pdm_df_output_gain_t gain
Decimation Filter Output Gain

struct __pdm_config
#include <fsl_pdm.h> PDM user configuration structure.

Public Members

bool enableDoze
This module will enter disable/low leakage mode if DOZEN is active with ipg_doze is asserted

bool enableFilterBypass
Switchable bypass path for the decimation filter

uint8_t fifoWatermark
Watermark value for FIFO

pdm_df_quality_mode_t qualityMode
Quality mode

uint8_t cicOverSampleRate
CIC filter over sampling rate

struct __pdm_transfer
#include <fsl_pdm.h> PDM SDMA transfer structure.

Public Members

volatile uint8_t *data
Data start address to transfer.

volatile size_t dataSize
Total Transfer bytes size.

struct __pdm_handle
#include <fsl_pdm.h> PDM handle structure.

Public Members

uint32_t state
: Transfer status

pdm_transfer_callback_t callback
: Callback function called at transfer event

void *userData
: Callback parameter passed to callback function

pdm_transfer_t pdmQueue[(4U)]
: Transfer queue storing queued transfer

size_t transferSize[(4U)]
: Data bytes need to transfer

volatile uint8_t queueUser
: Index for user to queue transfer

volatile uint8_t queueDriver
: Index for driver to get the transfer data and size

uint32_t format
: data format

uint8_t watermark
: Watermark value

uint8_t startChannel
: end channel

uint8_t channelNums
: Enabled channel number

2.69 PDM EDMA Driver

void PDM_TransferInstallEDMATCDMemory(*pdm_edma_handle_t* *handle, void *tcdAddr, size_t tcdNum)

Install EDMA descriptor memory.

Parameters

- handle – Pointer to EDMA channel transfer handle.
- tcdAddr – EDMA head descriptor address.
- tcdNum – EDMA link descriptor address.

void PDM_TransferCreateHandleEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle, *pdm_edma_callback_t* callback, void *userData, *edma_handle_t* *dmaHandle)

Initializes the PDM Rx eDMA handle.

This function initializes the PDM slave DMA handle, which can be used for other PDM master transactional APIs. Usually, for a specified PDM instance, call this API once to get the initialized handle.

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- dmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void PDM_TransferSetMultiChannelInterleaveType(*pdm_edma_handle_t* *handle, *pdm_edma_multi_channel_interleave_t* multiChannelInterleaveType)

Initializes the multi PDM channel interleave type.

This function initializes the PDM DMA handle member interleaveType, it shall be called only when application would like to use type kPDM_EDMAMultiChannelInterleavePerChannelBlock, since the default interleaveType is kPDM_EDMAMultiChannelInterleavePerChannelSample always

Parameters

- handle – PDM eDMA handle pointer.
- multiChannelInterleaveType – Multi channel interleave type.

void PDM_TransferSetChannelConfigEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle, uint32_t channel, const *pdm_channel_config_t* *config)

Configures the PDM channel.

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.
- channel – channel index.
- config – pdm channel configurations.

status_t PDM_TransferReceiveEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle, *pdm_edma_transfer_t* *xfer)

Performs a non-blocking PDM receive using eDMA.

Mcaro MCUX_SDK_PDM_EDMA_PDM_ENABLE_INTERNAL can control whether PDM is enabled internally or externally.

a. Scatter gather case: This functio support dynamic scatter gather and staic scatter gather, a. for the dynamic scatter gather case: Application should call PDM_TransferReceiveEDMA function continuously to make sure new receive request is submit before the previous one finish. b. for the static scatter gather case: Application should use the link transfer feature and make sure a loop link transfer is provided, such as:

```
pdm_edma_transfer_t pdmXfer[2] =
{
    {
    .data  = s_buffer,
    .dataSize = BUFFER_SIZE,
    .linkTransfer = &pdmXfer[1],
    },

    {
    .data  = &s_buffer[BUFFER_SIZE],
    .dataSize = BUFFER_SIZE,
    .linkTransfer = &pdmXfer[0]
    },
};
```

b. Multi channel case: This function support receive multi pdm channel data, for example, if two channel is requested,

```
PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &s_pdmRxHandle_0, DEMO_PDM_
↪ENABLE_CHANNEL_0, &channelConfig);
PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &s_pdmRxHandle_0, DEMO_PDM_
↪ENABLE_CHANNEL_1, &channelConfig);
PDM_TransferReceiveEDMA(DEMO_PDM, &s_pdmRxHandle_0, pdmXfer);
```

The output data will be formatted as below if handle->interleaveType =

Note: This interface returns immediately after the transfer initiates. Call the PDM_GetReceiveRemainingBytes to poll the transfer status and check whether the PDM transfer is finished.

---

void PDM_TransferTerminateReceiveEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle)

Terminate all PDM receive.

This function will clear all transfer slots buffered in the pdm queue. If users only want to abort the current transfer slot, please call PDM_TransferAbortReceiveEDMA.

Parameters

- base – PDM base pointer.
- handle – PDM eDMA handle pointer.

void PDM_TransferAbortReceiveEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle)

Aborts a PDM receive using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call PDM_TransferTerminateReceiveEDMA.

Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.

status_t PDM_TransferGetReceiveCountEDMA(PDM_Type *base, *pdm_edma_handle_t* *handle, size_t *count)

Gets byte count received by PDM.

Parameters

- base – PDM base pointer
- handle – PDM eDMA handle pointer.

- count – Bytes count received by PDM.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

FSL_PDM_EDMA_DRIVER_VERSION

Version 2.6.5

enum _pdm_edma_multi_channel_interleave

pdm multi channel interleave type

Values:

enumerator kPDM_EDMAMultiChannelInterleavePerChannelSample

enumerator kPDM_EDMAMultiChannelInterleavePerChannelBlock

typedef struct *_pdm_edma_handle* pdm_edma_handle_t

PDM edma handler.

typedef enum *_pdm_edma_multi_channel_interleave* pdm_edma_multi_channel_interleave_t

pdm multi channel interleave type

typedef struct *_pdm_edma_transfer* pdm_edma_transfer_t

PDM edma transfer.

typedef void (*pdm_edma_callback_t)(PDM_Type *base, *pdm_edma_handle_t* *handle, *status_t* status, void *userData)

PDM eDMA transfer callback function for finish and error.

MCUX_SDK_PDM_EDMA_PDM_ENABLE_INTERNAL

the PDM enable position When calling PDM_TransferReceiveEDMA

struct _pdm_edma_transfer

#include <fsl_pdm_edma.h> PDM edma transfer.

Public Members

volatile uint8_t *data

Data start address to transfer.

volatile size_t dataSize

Total Transfer bytes size.

struct *_pdm_edma_transfer* *linkTransfer

linked transfer configurations

struct __pdm__edma__handle

#include <fsl_pdm_edma.h> PDM DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for PDM send

uint8_t count

The transfer data count in a DMA request

uint32_t receivedBytes

total transfer count

uint32_t state

Internal state for PDM eDMA transfer

pdm_edma_callback_t callback

Callback for users while transfer finish or error occurs

bool isLoopTransfer

loop transfer

void *userData

User callback parameter

edma_tcd_t *tcd

TCD pool for eDMA transfer.

uint32_t tcdNum

TCD number

uint32_t tcdUser

Index for user to queue transfer.

uint32_t tcdDriver

Index for driver to get the transfer data and size

volatile uint32_t tcdUsedNum

Index for user to queue transfer.

pdm_edma_multi_channel_interleave_t interleaveType

multi channel transfer interleave type

uint8_t endChannel

The last enabled channel

uint8_t channelNums
total channel numbers

2.70 PINT: Pin Interrupt and Pattern Match Driver

FSL_PINT_DRIVER_VERSION

enum _pint_pin_enable
PINT Pin Interrupt enable type.
Values:

enumerator kPINT_PinIntEnableNone
Do not generate Pin Interrupt

enumerator kPINT_PinIntEnableRiseEdge
Generate Pin Interrupt on rising edge

enumerator kPINT_PinIntEnableFallEdge
Generate Pin Interrupt on falling edge

enumerator kPINT_PinIntEnableBothEdges
Generate Pin Interrupt on both edges

enumerator kPINT_PinIntEnableLowLevel
Generate Pin Interrupt on low level

enumerator kPINT_PinIntEnableHighLevel
Generate Pin Interrupt on high level

enum _pint_int
PINT Pin Interrupt type.
Values:

enumerator kPINT_PinInt0
Pin Interrupt 0

enum _pint_pmatch_input_src
PINT Pattern Match bit slice input source type.
Values:

enumerator kPINT_PatternMatchInp0Src
Input source 0

enumerator kPINT_PatternMatchInp1Src
Input source 1

enumerator kPINT_PatternMatchInp2Src
Input source 2

enumerator kPINT_PatternMatchInp3Src
Input source 3

enumerator kPINT_PatternMatchInp4Src
Input source 4

enumerator kPINT_PatternMatchInp5Src
Input source 5

enumerator kPINT_PatternMatchInp6Src
Input source 6

enumerator kPINT_PatternMatchInp7Src
Input source 7

enumerator kPINT_SecPatternMatchInp0Src
Input source 0

enumerator kPINT_SecPatternMatchInp1Src
Input source 1

enum _pint_pmatch_bslice
PINT Pattern Match bit slice type.
Values:

enumerator kPINT_PatternMatchBSlice0
Bit slice 0

enum _pint_pmatch_bslice_cfg
PINT Pattern Match configuration type.
Values:

enumerator kPINT_PatternMatchAlways
Always Contributes to product term match

enumerator kPINT_PatternMatchStickyRise
Sticky Rising edge

enumerator kPINT_PatternMatchStickyFall
Sticky Falling edge

enumerator kPINT_PatternMatchStickyBothEdges
Sticky Rising or Falling edge

enumerator kPINT_PatternMatchHigh
High level

enumerator kPINT_PatternMatchLow
Low level

enumerator kPINT_PatternMatchNever
Never contributes to product term match

enumerator kPINT_PatternMatchBothEdges
Either rising or falling edge

typedef enum *_pint_pin_enable* pint_pin_enable_t
PINT Pin Interrupt enable type.

typedef enum *_pint_int* pint_pin_int_t
PINT Pin Interrupt type.

typedef enum *_pint_pmatch_input_src* pint_pmatch_input_src_t
PINT Pattern Match bit slice input source type.

typedef enum *_pint_pmatch_bslice* pint_pmatch_bslice_t
PINT Pattern Match bit slice type.

typedef enum *_pint_pmatch_bslice_cfg* pint_pmatch_bslice_cfg_t
PINT Pattern Match configuration type.

typedef struct *_pint_status* pint_status_t
PINT event status.

typedef void (*pint_cb_t)(*pint_pin_int_t* pintr, *pint_status_t* *status)
PINT Callback function.

typedef struct *_pint_pmatch_cfg* pint_pmatch_cfg_t

void PINT_Init(PINT_Type *base)
Initialize PINT peripheral.

This function initializes the PINT peripheral and enables the clock.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

void PINT_SetCallback(PINT_Type *base, *pint_cb_t* callback)
Set PINT callback.

This function set the callback for PINT interupt handler.

Parameters

- base – Base address of the PINT peripheral.

- callback – Callback.

Return values
None. –

void PINT_PinInterruptConfig(PINT_Type *base, *pint_pin_int_t* intr, *pint_pin_enable_t* enable)

Configure PINT peripheral pin interrupt.

This function configures a given pin interrupt.

Parameters

- base – Base address of the PINT peripheral.
- intr – Pin interrupt.
- enable – Selects detection logic.

Return values
None. –

void PINT_PinInterruptGetConfig(PINT_Type *base, *pint_pin_int_t* pintr, *pint_pin_enable_t* *enable)

Get PINT peripheral pin interrupt configuration.

This function returns the configuration of a given pin interrupt.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.
- enable – Pointer to store the detection logic.

Return values
None. –

void PINT_PinInterruptClrStatus(PINT_Type *base, *pint_pin_int_t* pintr)

Clear Selected pin interrupt status only when the pin was triggered by edge-sensitive.

This function clears the selected pin interrupt status.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetStatus(PINT_Type *base, *pint_pin_int_t* pintr)

Get Selected pin interrupt status.

This function returns the selected pin interrupt status.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
status – = 0 No pin interrupt request. = 1 Selected Pin interrupt request active.

void PINT_PinInterruptClrStatusAll(PINT_Type *base)

Clear all pin interrupts status only when pins were triggered by edge-sensitive.

This function clears the status of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetStatusAll(PINT_Type *base)

Get all pin interrupts status.

This function returns the status of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values
status – Each bit position indicates the status of corresponding pin interrupt. = 0 No pin interrupt request. = 1 Pin interrupt request active.

static inline void PINT_PinInterruptClrFallFlag(PINT_Type *base, *pint_pin_int_t* pintr)

Clear Selected pin interrupt fall flag.

This function clears the selected pin interrupt fall flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetFallFlag(PINT_Type *base, *pint_pin_int_t* pintr)

Get selected pin interrupt fall flag.

This function returns the selected pin interrupt fall flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values
flag – = 0 Falling edge has not been detected. = 1 Falling edge has been detected.

static inline void PINT_PinInterruptClrFallFlagAll(PINT_Type *base)

Clear all pin interrupt fall flags.

This function clears the fall flag for all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

static inline uint32_t PINT_PinInterruptGetFallFlagAll(PINT_Type *base)

Get all pin interrupt fall flags.

This function returns the fall flag of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values
flags – Each bit position indicates the falling edge detection of the corresponding pin interrupt. 0 Falling edge has not been detected. = 1 Falling edge has been detected.

static inline void PINT_PinInterruptClrRiseFlag(PINT_Type *base, *pint_pin_int_t* pintr)

Clear Selected pin interrupt rise flag.

This function clears the selected pin interrupt rise flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values

None. –

static inline uint32_t PINT_PinInterruptGetRiseFlag(PINT_Type *base, *pint_pin_int_t* pintr)

Get selected pin interrupt rise flag.

This function returns the selected pin interrupt rise flag.

Parameters

- base – Base address of the PINT peripheral.
- pintr – Pin interrupt.

Return values

flag – = 0 Rising edge has not been detected. = 1 Rising edge has been detected.

static inline void PINT_PinInterruptClrRiseFlagAll(PINT_Type *base)

Clear all pin interrupt rise flags.

This function clears the rise flag for all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

static inline uint32_t PINT_PinInterruptGetRiseFlagAll(PINT_Type *base)

Get all pin interrupt rise flags.

This function returns the rise flag of all pin interrupts.

Parameters

- base – Base address of the PINT peripheral.

Return values

flags – Each bit position indicates the rising edge detection of the corresponding pin interrupt. 0 Rising edge has not been detected. = 1 Rising edge has been detected.

void PINT_PatternMatchConfig(PINT_Type *base, *pint_pmatch_bslice_t* bslice, *pint_pmatch_cfg_t* *cfg)

Configure PINT pattern match.

This function configures a given pattern match bit slice.

Parameters

- base – Base address of the PINT peripheral.
- bslice – Pattern match bit slice number.
- cfg – Pointer to bit slice configuration.

Return values

None. –

void PINT_PatternMatchGetConfig(PINT_Type *base, *pint_pmatch_bslice_t* bslice, *pint_pmatch_cfg_t* *cfg)

Get PINT pattern match configuration.

This function returns the configuration of a given pattern match bit slice.

Parameters

- base – Base address of the PINT peripheral.
- bslice – Pattern match bit slice number.
- cfg – Pointer to bit slice configuration.

Return values

None. –

static inline uint32_t PINT_PatternMatchGetStatus(PINT_Type *base, *pint_pmatch_bslice_t* bslice)

Get pattern match bit slice status.

This function returns the status of selected bit slice.

Parameters

- base – Base address of the PINT peripheral.
- bslice – Pattern match bit slice number.

Return values

status – = 0 Match has not been detected. = 1 Match has been detected.

static inline uint32_t PINT_PatternMatchGetStatusAll(PINT_Type *base)

Get status of all pattern match bit slices.

This function returns the status of all bit slices.

Parameters

- base – Base address of the PINT peripheral.

Return values

status – Each bit position indicates the match status of corresponding bit slice. = 0 Match has not been detected. = 1 Match has been detected.

uint32_t PINT_PatternMatchResetDetectLogic(PINT_Type *base)

Reset pattern match detection logic.

This function resets the pattern match detection logic if any of the product term is matching.

Parameters

- base – Base address of the PINT peripheral.

Return values

pmstatus – Each bit position indicates the match status of corresponding bit slice. = 0 Match was detected. = 1 Match was not detected.

static inline void PINT_PatternMatchEnable(PINT_Type *base)

Enable pattern match function.

This function enables the pattern match function.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

static inline void PINT_PatternMatchDisable(PINT_Type *base)

Disable pattern match function.

This function disables the pattern match function.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

static inline void PINT_PatternMatchEnableRXEV(PINT_Type *base)

Enable RXEV output.

This function enables the pattern match RXEV output.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

static inline void PINT_PatternMatchDisableRXEV(PINT_Type *base)

Disable RXEV output.

This function disables the pattern match RXEV output.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

void PINT_EnableCallback(PINT_Type *base)

Enable callback.

This function enables the interrupt for the selected PINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

- base – Base address of the PINT peripheral.

Return values

None. –

void PINT_DisableCallback(PINT_Type *base)

Disable callback.

This function disables the interrupt for the selected PINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

- base – Base address of the peripheral.

Return values

None. –

void PINT_Deinit(PINT_Type *base)

Deinitialize PINT peripheral.

This function disables the PINT clock.

Parameters

- base – Base address of the PINT peripheral.

Return values
None. –

void PINT_EnableCallbackByIndex(PINT_Type *base, *pint_pin_int_t* pintIdx)

enable callback by pin index.

This function enables callback by pin index instead of enabling all pins.

Parameters

- base – Base address of the peripheral.
- pintIdx – pin index.

Return values
None. –

void PINT_EnableInterruptByIndex(PINT_Type *base, *pint_pin_int_t* pintIdx)

enable interrupt in NVIC by pin index.

This function enables the interrupt in the NVIC. The difference with *PINT_EnableCallbackByIndex()* is that *PINT_EnableCallbackByIndex()* not only enables the interrupt in the NVIC but also clears pending interrupts. Use this function together with *PINT_DisableInterruptByIndex()* to temporarily disable/enable the pin interrupt. Use *PINT_EnableCallbackByIndex()* to enable the interrupt after installing the callback.

Parameters

- base – Base address of the peripheral.
- pinIdx – pin index.

Return values
None. –

void PINT_DisableInterruptByIndex(PINT_Type *base, *pint_pin_int_t* pintIdx)

disable interrupt in NVIC by pin index.

This function disables the interrupt in the NVIC. The difference with *PINT_DisableCallbackByIndex()* is that *PINT_DisableCallbackByIndex()* not only disables the interrupt in the NVIC but also clears pending interrupts. Use this function together with *PINT_EnableInterruptByIndex()* to temporarily disable/enable the pin interrupt. Use *PINT_DisableCallbackByIndex()* to disable the interrupt in a de-init function.

Parameters

- base – Base address of the peripheral.
- pinIdx – pin index.

Return values
None. –

void PINT_DisableCallbackByIndex(PINT_Type *base, *pint_pin_int_t* pintIdx)

disable callback by pin index.

This function disables callback by pin index instead of disabling all pins.

Parameters

- base – Base address of the peripheral.
- pintIdx – pin index.

Return values
None. –

void PINT_DriverIRQHandler(uint32_t instance)
PINT parameterized IRQ handler entry.

Parameters

- instance – PINT instance index.

PINT_USE_LEGACY_CALLBACK

PININT_BITSLICE_SRC_START

PININT_BITSLICE_SRC_MASK

PININT_BITSLICE_CFG_START

PININT_BITSLICE_CFG_MASK

PININT_BITSLICE_ENDP_MASK

PINT_PIN_INT_LEVEL

PINT_PIN_INT_EDGE

PINT_PIN_INT_FALL_OR_HIGH_LEVEL

PINT_PIN_INT_RISE

PINT_PIN_RISE_EDGE

PINT_PIN_FALL_EDGE

PINT_PIN_BOTH_EDGE

PINT_PIN_LOW_LEVEL

PINT_PIN_HIGH_LEVEL

struct _pint_status
#include <fsl_pint.h> PINT event status.

struct _pint_pmatch_cfg
#include <fsl_pint.h>

2.71 PNGDEC: PNG decoder Driver

Error codes for the PNGDEC driver.

Values:

enumerator kStatus_PNGDEC_NotSupported
Interlace data chunk is not supported.

enumerator kStatus_PNGDEC_WidthTooLarge
The image width is larger than 1024.

enum _pngdec_interrupt_enable
Interrupt enable/disable mask.

Values:

enumerator kPNGDEC_ChecksumErrorFlag
ADLER-32 checksum error status and interrupt enable.

enumerator kPNGDEC_CrcErrorFlag
CRC error status and interrupt enable.

enumerator kPNGDEC_HeaderErrorFlag
Header error status and interrupt enable.

enumerator kPNGDEC_BtypeErrorFlag
B type error status and interrupt enable. TODO

enumerator kPNGDEC_ZlibHeaderErrorFlag
Zlib header error status and interrupt enable.

enumerator kPNGDEC_BitDepthErrorFlag
Bit depth error status and interrupt enable. Bit depth less than 8 bit not supported.

enumerator kPNGDEC_InterlaceErrorFlag
Interlace error status and interrupt enable. Interlace is not supported.

enumerator kPNGDEC_WidthErrorFlag
Width error status and interrupt enable. Max image width 1024.

enumerator kPNGDEC_EncodeDataDoneFlag
Data encoding done status and interrupt enable.

enumerator kPNGDEC_DecodePixelDoneFlag
Pixel data decoding done status and interrupt enable.

enumerator kPNGDEC_DecodeAncillaryDoneFlag
Ancillary data decoding done status and interrupt enable.

enumerator kPNGDEC_ErrorFlags

enum _pngdec_endian_mode

PNG decode endian mode.

Values:

enumerator kPNGDEC_LittleEndian

Little endian.

enumerator kPNGDEC_BigEndian

Big endian.

enum _pngdec_color_type

PNG color type.

Values:

enumerator kPNGDEC_PixelFormatY8

8-bit monochrome(gray scale).

enumerator kPNGDEC_PixelFormatYA88

16-bit monochrome(gray scale) with alpha.

enumerator kPNGDEC_PixelFormatRGB888

24-bit RGB true color, 8-bit each palette.

enumerator kPNGDEC_PixelFormatRGBA8888

32-bit RGBA true color with alpha, 8-bit each palette.

enumerator kPNGDEC_PixelFormatRGB16_16_16

48-bit RGB true color, 16-bit each palette.

enumerator kPNGDEC_PixelFormatRGBA16_16_16_16

64-bit RGBA true color with alpha, 16-bit each palette.

enumerator kPNGDEC_PixelFormatLUT8

8-bit Indexed Color.

typedef enum *_pngdec_endian_mode* pngdec_endian_mode_t

PNG decode endian mode.

typedef struct *_pngdec_config* pngdec_config_t

Configuration for PNGDEC.

typedef enum *_pngdec_color_type* pngdec_color_type_t

PNG color type.

typedef struct *_pngdec_image* pngdec_image_t

The decoded image info structure.

FSL_PNGDEC_DRIVER_VERSION

PNGDEC driver version.

void PNGDEC_Init(PNGDEC_Type *base, const *pngdec_config_t* *config)

Initializes the PNGDEC.

The default configuration can be got by calling *PNGDEC_GetDefaultConfig()*.

Parameters

- base – PNGDEC peripheral base address.
- config – Pointer to PNGDEC configuration structure.

void PNGDEC_Deinit(PNGDEC_Type *base)

Deinitializes the PNGDEC.

Parameters

- base – PNGDEC peripheral base address.

void PNGDEC_GetDefaultConfig(*pngdec_config_t* *config)

Gets the default configuration for PNGDEC.

This function initializes the user configuration structure to default value. The default value are:

Example:

```
config->enableAncillary = false;
config->enable = false;
```

Parameters

- config – Pointer to PNGDEC configuration structure.

static inline void PNGDEC_EnableInterrupts(PNGDEC_Type *base, uint32_t mask)

Enables interrupts.

Parameters

- base – PNGDEC peripheral base address.
- mask – Interrupts mask. See "_pngdec_interrupt_enable".

static inline void PNGDEC_DisableInterrupts(PNGDEC_Type *base, uint32_t mask)

Disables interrupts.

Parameters

- base – PNGDEC peripheral base address.
- mask – Interrupts mask. See "_pngdec_interrupt_enable".

static inline uint32_t PNGDEC_GetStatusFlags(PNGDEC_Type *base)

Gets status flags.

Parameters

- base – PNGDEC peripheral base address.

Returns

Status flags asserted mask. See "_pngdec_status_flags".

static inline void PNGDEC_ClearStatusFlags(PNGDEC_Type *base, uint32_t mask)

Clears status flags.

Parameters

- base – PNGDEC peripheral base address.
- mask – Status flags mask. See "_pngdec_status_flags".

static inline void PNGDEC_Reset(PNGDEC_Type *base)

Resets the PNGDEC code and DMA logic.

Parameters

- base – PNGDEC peripheral base address.

static inline void PNGDEC_Enable(PNGDEC_Type *base, bool enable)

Enables or disables the PNGDEC.

Parameters

- base – PNGDEC peripheral base address.
- enable – True to enable the PNGDEC.

static inline void PNGDEC_EnableAncillary(PNGDEC_Type *base, bool enable)

Enables or disables the PNGDEC ancillary drop.

Parameters

- base – PNGDEC peripheral base address.
- enable – True to enable the PNGDEC ancillary drop.

static inline void PNGDEC_StartDecode(PNGDEC_Type *base)

Starts the PNG decode.

Parameters

- base – PNGDEC peripheral base address.

void PNGDEC_SetPngBuffer(PNGDEC_Type *base, uint8_t *buffer, size_t length)

Sets the address and length of the raw PNG image.

Parameters

- base – PNGDEC peripheral base address.
- data – Start address of the buffer of the raw PNG image, shall be 8-byte aligned.
- length – Size of the buffer.

static inline void PNGDEC_SetOutputBuffer(PNGDEC_Type *base, uint8_t *imageData, uint8_t *ancillaryData)

Sets the address of the decoded data.

Parameters

- base – PNGDEC peripheral base address.
- imageData – Start address of the buffer for the decoded image data, shall be 8-byte aligned.
- ancillaryData – Start address of the buffer for the decoded ancillary data, shall be 8-byte aligned.

status_t PNGDEC_ParseHeader(*pngdec_image_t* *image, uint8_t *pngBuf)

Parses the PNG header and stores the info in the decoded image info structure.

Parameters

- image – Pointer to PNGDEC decoded image info structure.
- pngBuf – Pointer to PNG file buffer.

Return values

- kStatus_Success – Header parsing success.
- kStatus_Fail – PNG header parsing failed due corrupted header.
- kStatus_PNGDEC_WidthTooLarge – Header parsing failed due to the image width is larger than 1024.
- kStatus_PNGDEC_NotSupported – Header parsing failed due to the image is interlaced or the bit depth is less than 8.

status_t PNGDEC_Decode(PNGDEC_Type *base, *pngdec_image_t* *image, uint32_t *status)

Decodes the PNG image.

This function performs the PNG decoding in blocking way and stores the decoded info in decoded image info structure.

Parameters

- base – PNGDEC peripheral base address.
- image – Pointer to PNGDEC decoded image info structure.
- status – Pointer to decoded status. When retval is kStatus_Fail, Checksum/Crc/Header/Btype/ZlibHeader error may occur due to PNG file corruption, user can check which error(s) occured if necessary.

Return values

- kStatus_Success – PNG decoding success.
- kStatus_Fail – PNG decoding failed due to CRC/header/B type/Alder error or invalid PNG file.
- kStatus_PNGDEC_WidthTooLarge – PNG decoding failed due to the image width is larger than 1024.
- kStatus_PNGDEC_NotSupported – PNG decoding failed due to the image is interlaced or the bit depth is less than 8.

bool enableAncillary

Enable ancillary data drop.

bool enable

Enables the decode IP.

uint32_t width

Decoded image width.

uint32_t height

Decoded image height.

pngdec_color_type_t colorType

Color type for the generated image.

uint8_t *imageData
Pointer to the buffer for decoded image data.

uint8_t *ancillaryData
Pointer to the buffer for decoded ancillary data.

uint32_t ancillaryLength
Length of the ancillary data.

struct __pngdec__config
#include <fsl_pngdec.h> Configuration for PNGDEC.

struct __pngdec__image
#include <fsl_pngdec.h> The decoded image info structure.

2.72 Power Driver

enum pd__bits
Power down control bits defined in SLEEPCON and PMC.
Values:

enumerator kPDRUNCFG__SHUT__SENSEP__MAINCLK
The Private Part in VDD1_SENSE Main Clock Shut Off.

enumerator kPDRUNCFG__SHUT__SENSES__MAINCLK
The Shared Part in VDD1_SENSE Main Clock Shut Off.

enumerator kPDRUNCFG__SHUT__RAM0__CLK
Main Clock of RAM Arbiter 0 Shut Off.

enumerator kPDRUNCFG__SHUT__RAM1__CLK
Main Clock of RAM Arbiter 1 shut off.

enumerator kPDRUNCFG__SHUT__COMNN__MAINCLK
VDDN_COM Domain Main Clock Shut Off.

enumerator kPDRUNCFG__SHUT__MEDIA__MAINCLK
VDD2_MEDIA and VDDN_MEDIA Domains Main Clock of Shut Off.

enumerator kPDRUNCFG__PD__SYSXTAL
Xtal Oscillator Power Down.

enumerator kPDRUNCFG__PD__FRO2
FRO2 Power Down.

enumerator kPDRUNCFG__PD__LPOSC
LPOSC Power Down.

enumerator kPDRUNCFG_PD_PLLANA
Main PLL Analog Function Power Down.

enumerator kPDRUNCFG_PD_PLLLDO
Main PLL Internal Regulator Power Down.

enumerator kPDRUNCFG_PD_AUDPLLANA
Audio PLL Analog Function Power Down.

enumerator kPDRUNCFG_PD_AUDPLLLDO
Audio PLL Internal Regulator Power Down.

enumerator kPDRUNCFG_PD_ADC0
ADC0 Power Down, 1 - Applies gating.

enumerator kPDRUNCFG_GATE_FRO2
FRO2 Gating, 1 - Applies gating.

enumerator kPDRUNCFG_DSR_VDD2N_MEDIA
Power Switch and DSR Enable for the VDD2_MEDIA and VDDN_MEDIA domains.

enumerator kPDRUNCFG_DSR_VDDN_COM
Power Switch and DSR Enable for the VDDN_COM domain.

enumerator kPDRUNCFG_PD_VDD2_DSP
Power switch for the HiFi4 DSP.

enumerator kPDRUNCFG_PD_VDD2_MIPI
Power Switch for the MIPI PHY.

enumerator kPDRUNCFG_PD_RBB_VDD1
Power Down RBB in VDD1.

enumerator kPDRUNCFG_PD_AFBB_VDD1
Power Down AFBB in VDD1 Domain.

enumerator kPDRUNCFG_PD_RBB_VDD2
Power Down RBB in VDD2 Domain.

enumerator kPDRUNCFG_PD_AFBB_VDD2
Power Down AFBB in VDD2 Domain.

enumerator kPDRUNCFG_PD_RBB_VDDN
Power Down RBB in VDDN Domain.

enumerator kPDRUNCFG_PD_AFBB_VDDN
Power Down RBB in VDDN Domain.

enumerator kPDRUNCFG_PD_SRAM_RBB_VDD1
Power Down SRAM RBB in VDD1 Domain.

enumerator kPDRUNCFG_PD_SRAM_RBB_VDD2
Power Down SRAM RBB in VDD2 Domain.

enumerator kPDRUNCFG_PD_SRAM_AFBB_VDD2
Power Down SRAM AFBB in VDD2 Domain.

enumerator kPDRUNCFG_PD_PMC_TEMPSNS
Power Down PMC Temperature Sensor.

enumerator kPDRUNCFG_LP_PMCREF
PMC References in Low-Power Mode.

enumerator kPDRUNCFG_PD_HVD1V8
HVD Power Down.

enumerator kPDRUNCFG_LP_POR_VDD1
POR Low-Power Mode in VDD1 Domain.

enumerator kPDRUNCFG_LP_LVD_VDD1
LVD Low-Power Mode in VDD1 Domain.

enumerator kPDRUNCFG_PD_HVD_VDD1
Power Down HVD in VDD1 Domain.

enumerator kPDRUNCFG_PD_AGDET_VDD1
Power Down AGDET in VDD1 Domain.

enumerator kPDRUNCFG_LP_POR_VDD2
POR Low-Power Mode in VDD2 Domain.

enumerator kPDRUNCFG_LP_LVD_VDD2
LVD Low-Power Mode in VDD2 Domain.

enumerator kPDRUNCFG_PD_HVD_VDD2
Power Down HVD in VDD2 Domain.

enumerator kPDRUNCFG_PD_AGDET_VDD2
Power Down AGDET in VDD2 Domain.

enumerator kPDRUNCFG_LP_POR_VDDN
POR Low-Power Mode in VDDN Domain.

enumerator kPDRUNCFG_LP_LVD_VDDN
LVD Low-Power Mode in VDDN Domain.

enumerator kPDRUNCFG_PD_HVD_VDDN
Power Down HVD in VDDN Domain.

enumerator kPDRUNCFG_PD_OTP
Power Down OTP.

enumerator kPDRUNCFG_PD_ROM
Power Down ROM.

enumerator kPDRUNCFG_APD_SRAM1
RAM Partition 1 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM2
RAM Partition 2 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM3
RAM Partition 3 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM4
RAM Partition 4 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM5
RAM Partition 5 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM6
RAM Partition 6 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM7
RAM Partition 7 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM8
RAM Partition 8 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM9
RAM Partition 9 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM10
RAM Partition 10 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM11
RAM Partition 11 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM12
RAM Partition 12 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM13
RAM Partition 13 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM14
RAM Partition 14 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM15
RAM Partition 15 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM16
RAM Partition 16 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM17
RAM Partition 17 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM18
RAM Partition 18 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM19
RAM Partition 19 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM20
RAM Partition 20 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM21
RAM Partition 21 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM22
RAM Partition 22 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM23
RAM Partition 23 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM24
RAM Partition 24 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM25
RAM Partition 25 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM26
RAM Partition 26 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM27
RAM Partition 27 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM28
RAM Partition 28 Array Power Down.

enumerator kPDRUNCFG_APD_SRAM29
RAM Partition 29 Array Power Down.

enumerator kPDRUNCFG_PPD_SRAM1
RAM Partition 1 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM2
RAM Partition 2 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM3
RAM Partition 3 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM4
RAM Partition 4 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM5
RAM Partition 5 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM6
RAM Partition 6 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM7
RAM Partition 7 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM8
RAM Partition 8 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM9
RAM Partition 9 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM10
RAM Partition 10 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM11
RAM Partition 11 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM12
RAM Partition 12 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM13
RAM Partition 13 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM14
RAM Partition 14 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM15
RAM Partition 15 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM16
RAM Partition 16 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM17
RAM Partition 17 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM18
RAM Partition 18 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM19
RAM Partition 19 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM20
RAM Partition 20 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM21
RAM Partition 21 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM22
RAM Partition 22 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM23
RAM Partition 23 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM24
RAM Partition 24 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM25
RAM Partition 25 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM26
RAM Partition 26 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM27
RAM Partition 27 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM28
RAM Partition 28 Periphery Power Down.

enumerator kPDRUNCFG_PPD_SRAM29
RAM Partition 29 Periphery Power Down.

enumerator kPDRUNCFG_APD_SDHC0_SRAM
uSDHC0 SRAM array Power Down.

enumerator kPDRUNCFG_APD_SDHC1_SRAM
uSDHC1 SRAM array Power Down.

enumerator kPDRUNCFG_APD_USB0_SRAM
USB0 SRAM array Power Down.

enumerator kPDRUNCFG_APD_USB1_SRAM
USB1 SRAM array Power Down.

enumerator kPDRUNCFG_APD_JPEGDEC
Powers down JPEGDEC RAMs array.

enumerator kPDRUNCFG_APD_PNGDEC
Powers down PNGDEC RAMs array.

enumerator kPDRUNCFG_APD_MIPIDSI
Powers down MIPI PHY RAMs array.

enumerator kPDRUNCFG_APD_GPU
Powers down VGPU RAMs array.

enumerator kPDRUNCFG_APD_DMA2_3
Powers down DMA2 and DMA3 RAMs array.

enumerator kPDRUNCFG_APD_DMA0_1_PKC_ETF
Powers down DMA0-1, PKC, and ETF RAMs array.

enumerator kPDRUNCFG_APD_CPU0_CODE_CACHE
Powers down CPU0 code cache RAMs array.

enumerator kPDRUNCFG_APD_CPU0_SYSTEM_CACHE
Powers down CPU0 system cache RAMs array.

enumerator kPDRUNCFG_APD_DSP_ICACHE
Powers down HiFi4 instruction cache RAMs array.

enumerator kPDRUNCFG_APD_DSP_DCACHE
Powers down HiFi4 data cache RAMs array.

enumerator kPDRUNCFG_APD_DSP_ITCM
Powers down HiFi4 instruction TCM RAMs array.

enumerator kPDRUNCFG_APD_DSP_DTCM
Powers down HiFi4 data TCM RAMs array.

enumerator kPDRUNCFG_APD_EZHV_TCM
Powers down EZH-V TCM RAMs array.

enumerator kPDRUNCFG_APD_NPU
Powers down NPU RAMs array.

enumerator kPDRUNCFG_APD_XSPI0
Powers down XSPI0, MMU0, and cache RAMs array.

enumerator kPDRUNCFG_APD_XSPI1
Powers down XSPI1, MMU1, and cache RAMs array.

enumerator kPDRUNCFG_APD_XSPI2
Powers down XSPI2 and MMU2 RAMs array.

enumerator kPDRUNCFG_APD_LCDIF
Powers down LCDIF controller RAMs array.

enumerator kPDRUNCFG_APD_OCOTP
Powers down OCOTP shadow RAMs array.

enumerator kPDRUNCFG_PPD_SDHC0_SRAM
Powers down uSDHC0 RAMs periphery.

enumerator kPDRUNCFG_PPD_SDHC1_SRAM
Powers down uSDHC1 RAMs periphery.

enumerator kPDRUNCFG_PPD_USB0_SRAM
Powers down USB0 RAMs periphery.

enumerator kPDRUNCFG_PPD_USB1_SRAM
Powers down USB1 RAMs periphery.

enumerator kPDRUNCFG_PPD_JPEGDEC
Powers down JPEGDEC RAMs periphery.

enumerator kPDRUNCFG_PPD_PNGDEC
Powers down PNGDEC RAMs periphery.

enumerator kPDRUNCFG_PPD_MIPIDSI
Powers down MIPI PHY RAMs periphery.

enumerator kPDRUNCFG_PPD_GPU
Powers down VGPU RAMs periphery.

enumerator kPDRUNCFG_PPD_DMA2_3
Powers down DMA2 and DMA3 RAMs periphery.

enumerator kPDRUNCFG_PPD_DMA0_1_PKC_ETF
Powers down DMA0-1, PKC, and ETF RAMs periphery.

enumerator kPDRUNCFG_PPD_CPU0_CODE_CACHE
Powers down CPU0 code cache RAMs periphery.

enumerator kPDRUNCFG_PPD_CPU0_SYSTEM_CACHE
Powers down CPU0 system cache RAMs periphery.

enumerator kPDRUNCFG_PPD_DSP_ICACHE
Powers down HiFi4 instruction cache RAMs periphery.

enumerator kPDRUNCFG_PPD_DSP_DCACHE
Powers down HiFi4 data cache RAMs periphery.

enumerator kPDRUNCFG_PPD_DSP_ITCM
Powers down HiFi4 instruction TCM RAMs periphery.

enumerator kPDRUNCFG_PPD_DSP_DTCM
Powers down HiFi4 data TCM RAMs periphery.

enumerator kPDRUNCFG_PPD_EZHV_TCM
Powers down EZH-V TCM RAMs periphery.

enumerator kPDRUNCFG_PPD_NPU
Powers down NPU RAMs periphery.

enumerator kPDRUNCFG_PPD_XSPI0
Powers down XSPI0, MMU0, and cache RAMs periphery.

enumerator kPDRUNCFG_PPD_XSPI1
Powers down XSPI1, MMU1, and cache RAMs periphery.

enumerator kPDRUNCFG_PPD_XSPI2
Powers down XSPI2 and MMU2 RAMs periphery.

enumerator kPDRUNCFG_PPD_LCDIF
Powers down LCDIF controller RAMs periphery.

enumerator kPDRUNCFG_PPD_OCOTP
Powers down OCOTP shadow RAMs periphery.

enumerator kPDRUNCFG_ForceUnsigned

enum lp_bits
low power control bits for modules. Bits used for modules to requesting low-power.
Values:

enumerator kPower_VGPU_LPREQ
VGPU low power request.

enumerator kPower_EZHV_STOP
EZHV stop request.

enumerator kPower_EZHV_HALT
EZHV halt request.

enumerator kPower_EZHV_EXIT_WAIT
EZHV exit wait request.

enumerator kPower_XSPI2_STOP
XSPI2 stop request.

enumerator kPower_LPSPI14_B_LPREQ
LPSPI14 bus low power request.

enumerator kPower_LPSPI14_F_LPREQ
LPSPI14 function low power request.

enumerator kPower_LPSPI16_B_LPREQ
LPSPI16 bus low power request.

enumerator kPower_LPSPI16_F_LPREQ
LPSPI16 function low power request.

enumerator kPower_FLEXIO_B_LPREQ
FLEXIO bus low power request.

enumerator kPower_FLEXIO_F_LPREQ
FLEXIO function low power request.

enumerator kPower_NIC0_LPREQ
NIC0 low power request.

enumerator kPower_NIC1_LPREQ
NIC1 low power request.

enumerator kPower_ADC0_STOP
ADC0 stop request.

enumerator kPower_ACMP0_STOP
ACMP0 stop request.

enumerator kPower_MICFIL_STOP
MICFIL stop request.

enumerator kPower_LPI2C15_B_LPREQ
LPI2C15 bus low power request.

enumerator kPower_LPI2C15_F_LPREQ
LPI2C15 function low power request.

enumerator kPower_GDET3_LPREQ
GDET3 low power request.

enumerator kPower_RTC_STOP
RTC0 and RTC1 stop request.

enumerator kPower_EDMA2_STOP
EDMA2 stop request.

enumerator kPower_EDMA3_STOP
EDMA3 stop request.

enumerator kPower_FC17_B_LPREQ
LP_FLEXCOMM17 bus low power request.

enumerator kPower_FC18_B_LPREQ
LP_FLEXCOMM18 bus low power request.

enumerator kPower_FC19_B_LPREQ
LP_FLEXCOMM19 bus low power request.

enumerator kPower_FC20_B_LPREQ
LP_FLEXCOMM20 bus low power request.

enumerator kPower_FC17_F_LPREQ
LP_FLEXCOMM17 function low power request.

enumerator kPower_FC18_F_LPREQ
LP_FLEXCOMM18 function low power request.

enumerator kPower_FC19_F_LPREQ
LP_FLEXCOMM19 function low power request.

enumerator kPower_FC20_F_LPREQ
LP_FLEXCOMM20 function low power request.

enumerator kPower_SAI3_LPREQ
SAI3 low power request.

enumerator kPower_GPIO8_LPREQ
GPIO8 low power request.

enumerator kPower_GPIO9_LPREQ
GPIO9 low power request.

enumerator kPower_GPIO10_LPREQ
GPIO10 low power request.

enumerator kPower_I3C2_STOP
I3C2 low power request.

enumerator kPower_I3C3_STOP
I3C3 low power request.

enum _pmic_mode_reg
PMIC mode pin configuration API parameter.
Values:

enumerator kCfg_Run
PDRUNCFG0 register offset.

enumerator kCfg_Sleep
PDSLEEPCFG0 register offset.

enum _power_mode_config
Power mode configuration API parameter.
Values:

enumerator kPower_Sleep
Sleep mode.

enumerator kPower_DeepSleep
Deep Sleep mode.

enumerator kPower_DeepSleepRetention
Deep Sleep Retention(DSR) mode.

enumerator kPower_DeepPowerDown
Deep Power Down(DPD) mode.

enumerator kPower_FullDeepPowerDown
Full Deep Power Down(FDPD) mode.

enum _power_hwwake_src
DMA Wakeup sources.
Values:

enumerator kPower_HWWakeSrcFlexio
Enables DMA to wakeup by FLEXIO DMA request.

enumerator kPower_HWWakeSrcLpFlexcomm
Enables DMA to wakeup by LP_FLEXCOMM0-13(CPU0), LP_FLEXCOMM17-20(CPU1) DMA request.

enumerator kPower_HWWakeSrcSai
Enables DMA to wakeup by SAI0-2(CPU0), SAI3(CPU1) request.

enum _power_shared_resource_mask

Definition for shared resources.

Values:

enumerator kPower_MaskAdc0

ADC0 mask.

enumerator kPower_MaskAcmp0

ACMP0 mask.

enumerator kPower_MaskMicfil

MICFIL mask.

enumerator kPower_MaskLpi2c15

LPI2C15 mask.

enumerator kPower_MaskGdet3

GDET3 mask.

enumerator kPower_MaskRtc

RTC mask.

enumerator kPower_MaskVgpu

VGPU mask.

enumerator kPower_MaskEzhv

EZHV mask.

enumerator kPower_MaskXspi2

XSPI2 mask.

enumerator kPower_MaskLpspi14

LPSPI14 mask.

enumerator kPower_MaskLpspi16

LPSPI16 mask.

enumerator kPower_MaskFlexio

FlexIO mask.

enumerator kPower_MaskNic0

NIC0 mask.

enumerator kPower_MaskNic1

NIC1 mask.

enumerator kPower_MaskAll

Mask all shared modules.

enum _power_pad_vrange_val

pad voltage range value for the 3V capable IOs. To reduce power the IOs can be configured for 1.8V or 3.3V operation, powering down the detector, but requiring the application to restrict the IO supply voltage to the selected range.

Values:

enumerator kPadVol_Continuous

Continuous mode, VDDE detector on.

enumerator kPadVol_171_198

Voltage from 1.71V to 1.98V. supply detector off.

enumerator kPadVol_300_360

Voltage from 3.00V to 3.60V. supply detector off.

enum _body_bias_domain

Body Bias domain definition.

Values:

enumerator kPower_BodyBiasVdd2Sram

VDD2 domain RAMs.

enumerator kPower_BodyBiasVdd1Sram

VDD1 domain RAMs.

enumerator kPower_BodyBiasVddn

VDDN domain.

enumerator kPower_BodyBiasVdd2

VDD2 domain.

enumerator kPower_BodyBiasVdd1

VDD1 domain.

enum _power_regulator

Regulator definition for Regulator and LVD configuration.

Values:

enumerator kRegulator_DCDC

DCDC used for VDDN.

enumerator kRegulator_Vdd2LDO

VDD2 LDO.

enumerator kRegulator_Vdd1LDO

VDD1 LDO.

enum _power_ldo_mode

LDO mode definition.

Values:

enumerator kPower_LDOMode_Bypass

LDO Bypass mode.

enumerator kPower_LDOMode_HP

LDO High Power mode.

enumerator kPower_LDOMode_LP

LDO Low Power mode.

enum _power_dcdc_mode

DCDC mode definition.

Values:

enumerator kPower_DCDCMode_HP

DCDC High Power mode.

enumerator kPower_DCDCMode_LP

DCDC Low Power mode.

enumerator kPower_DCDCMode_ULP

DCDC Ultra Low Power mode. A0 Silicon does not support this mode and will default to low power mode when configured.

enum _power_vdd_src

VDDN, VDD1 or VDD2 supply source.

Values:

enumerator kVddSrc_PMC

Supplied by onchip regulator in PMC, DCDC for VDDN, LDO1 for VDD1, LDO2 for VDD2.

enumerator kVddSrc_PMIC

Supplied by external PMIC.

typedef enum *pd_bits* pd_bit_t

Power down control bits defined in SLEEPCON and PMC.

typedef enum *lp_bits* lp_bit_t

low power control bits for modules. Bits used for modules to requesting low-power.

typedef enum *_pmic_mode_reg* pmic_mode_reg_t

PMIC mode pin configuration API parameter.

typedef enum *_power_mode_config* power_mode_cfg_t

Power mode configuration API parameter.

typedef enum *_power_pad_vrange_val* power_pad_vrange_val_t

pad voltage range value for the 3V capable IOs. To reduce power the IOs can be configured for 1.8V or 3.3V operation, powering down the detector, but requiring the application to restrict the IO supply voltage to the selected range.

typedef enum *_power_regulator* power_regulator_t

Regulator definition for Regulator and LVD configuration.

typedef union *_power_regulator_voltage* power_regulator_voltage_t

Regulator voltage configuraton. Configure the regulator voltage in uV.

typedef enum *_power_vdd_src* power_vdd_src_t

VDDN, VDD1 or VDD2 supply source.

typedef union *_power_lvd_voltage* power_lvd_voltage_t

LVD voltage configuraton. Configure the LVD voltage in uV. Falling trip = 0.5V + 10 mV * bitfield value.

void POWER_EnablePD(*pd_bit_t* en)

API to enable PDRUNCFG bit in the SLEEPCON and PMC. Note that enabling the bit powers down the peripheral.

Parameters

- en – peripheral for which to enable the PDRUNCFG bit

void POWER_DisablePD(*pd_bit_t* en)

API to disable PDRUNCFG bit in the SLEEPCON and PMC. Note that disabling the bit powers up the peripheral.

Parameters

- en – peripheral for which to disable the PDRUNCFG bit

status_t POWER_ModuleEnterLPRequest(*lp_bit_t* en)

API to request a module entering low power. This API send low power/stop requests to a module and wait the acknowledgement from the module. For peripherals that are being used by a domain, using this low power request handshake is recommended to make sure the module is fully stopped before the module is powered down or clock gated.

Parameters

- en – peripheral for which to enable the low power request.

Returns

kStatus_Success for succeed, kStatus_Timeout for tiemout.

void POWER_ModuleExitLPRequest(*lp_bit_t* en)

API to negate low power/stop requests to a module(module resume regular operation).

Parameters

- en – peripheral for which to disable the low power request.

void POWER_SetPmicMode(uint32_t mode, *pmic_mode_reg_t* reg)

Set PMIC_MODE pins configure value.

Parameters

- mode – : PMIC MODE pin value. This field is aggregated low.
- reg – : PDSLEEPCFG0 or PDRUNCFG0 register offset

Returns
PMIC_MODE pins value in PDSLEEPCFG0

uint32_t POWER_GetPmicMode(*pmic_mode_reg_t* reg)

Get PMIC_MODE pins configure value.

Parameters

- reg – : PDSLEEPCFG0 or PDRUNCFG0 register offset

Returns
PMIC_MODE pins value in PDSLEEPCFG0 or PDRUNCFG0

static inline void POWER_EnableDeepSleep(void)

API to enable deep sleep bit in the ARM Core.

static inline void POWER_DisableDeepSleep(void)

API to disable deep sleep bit in the ARM Core.

void POWER_ApplyPD(void)

API to apply updated PMC PDRUNCFG bits in the Sysctl0.

void POWER_ClearEventFlags(uint32_t statusMask)

Clears the PMC event flags state.

Parameters

- statusMask – : A bitmask of event flags that are to be cleared.

uint32_t POWER_GetEventFlags(void)

Get the PMC event flags state.

Returns
PMC FLAGS register value

void POWER_EnableInterrupts(uint32_t interruptMask)

Enable the PMC interrupt requests.

Parameters

- interruptMask – : A bitmask of interrupts to enable.

void POWER_DisableInterrupts(uint32_t interruptMask)

Disable the PMC interrupt requests.

Parameters

- interruptMask – : A bitmask of interrupts to disable.

void POWER_EnableResets(uint32_t resetMask)

Enable the PMC reset requests.

Parameters

- resetMask – : A bitmask of Reset to enable, refer to PMC CTRL register descritpion in RM.

void POWER_DisableResets(uint32_t resetMask)

Disable the PMC reset requests.

Parameters

- resetMask – : A bitmask of Reset to disable, refer to PMC CTRL register descritpion in RM.

void EnableDeepSleepIRQ(IRQn_Type interrupt)

Enable specific interrupt for wake-up from deep-sleep mode. Enable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

Note: This function also enables the interrupt in the NVIC (*EnableIRQ()* is called internally).

Parameters

- interrupt – The IRQ number.

void DisableDeepSleepIRQ(IRQn_Type interrupt)

Disable specific interrupt for wake-up from deep-sleep mode. Disable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

Note: This function also disables the interrupt in the NVIC (*DisableIRQ()* is called internally).

Parameters

- interrupt – The IRQ number.

void POWER_EnableDMAHWWake(uint32_t sources)

Enable specific DMA channels to generate a wake-up from deep-sleep mode. DMA wakeup is an automatic hardware mechanism that handles DMA transactions for selected modules without the need for core processing. Software can configure individual DMA channels to generate a wakeup when they receive a DMA request. The DMA wakeup triggers an exit from Deep Sleep mode except the core, which remains sleeping.

Parameters

- sources – The bits mask for eDMA wakeup sources. Refers *_power_hwwake_src*.

void POWER_DisableDMAHWWake(uint32_t sources)

Disable specific DMA channels to generate a wake-up from deep-sleep mode. Disable the DMA channels for wake-up from deep sleep mode.

Parameters

- sources – The bits mask for eDMA wakeup sources. Refers *_power_hwwake_src*.

void POWER_EnableAutoWake(uint16_t ticks)

Configure PMC for auto wakeup.This feature allows PMC to wake up some amount of time after the domain enters Deep Sleep. The timer is clocked by PMC's internal 16 MHz clock. If both timers in compute and sense are enabled at the same time, wake-up will occur after BOTH timers time out (i.e. the max timeout value between the two is used).

Parameters

- ticks – : The ticks of 16MHz PMC internal clock to wait before waking up the domain after it goes to DeepSleep.

void POWER_EnableLPRequestMask(uint32_t mask)

Enable low power request masking for shared modules. Enabling the mask will relinquish the caller domain's control over the module, and give the other domain exclusive control over low-power requests to the module. NOTE, The masking of a shared module must NOT be disabled in both Compute and Sense domain.

Parameters

- mask – An OR'd bitmask of shared modules to be masked from low power request, ref *_power_shared_resource_mask* and SLEEPCON SHARED_MASK0 register in the RM. DeepSleep.

void POWER_DisableLPRequestMask(uint32_t mask)

Disable low power request masking for shared modules. Disabling the mask means this domain will own the module. Only one domain should own each shared module.

Parameters

- mask – An OR'd bitmask of shared modules to be enabled from low power request, ref *_power_shared_resource_mask* and SLEEPCON SHARED_MASK0 register in the RM. DeepSleep.

void POWER_EnableRunAFBB(uint32_t mask)

Enable AFBB mode for various domains in active mode. Note, users should call *POWER_ApplyPD()* to make the change take effect.

Parameters

- mask – : A bitmask of domains to enable AFBB mode, refer to *_body_bias_domain* and PMC PDRUNCFG0 register descritpion in RM.

void POWER_EnableSleepAFBB(uint32_t mask)

Enable AFBB mode for various domains in deep sleep mode. Note, users should call *POWER_ApplyPD()* to make the change take effect.

Parameters

- mask – : A bitmask of domains to enable AFBB mode, refer to *_body_bias_domain* and PMC PDSLEEPCFG0 register descritpion in RM.

void POWER_EnableRunRBB(uint32_t mask)

Enable RBB mode for various domains in active mode. This function should only be used when one CPU will not be making use of one of the voltage domains.Configuring the unused voltages for RBB will allow the other domain to have full control of the body bias mode for the voltages even when both domains are in run mode. Note, users should call *POWER_ApplyPD()* to make the change take effect.

Parameters

- mask – : A bitmask of domains to enable RBB mode, refer to *_body_bias_domain* and PMC PDRUNCFG0 register descritpion in RM.

void POWER_EnableSleepRBB(uint32_t mask)

Enable RBB mode for various domains in deep sleep mode.

Parameters

- mask – : A bitmask of domains to enable RBB mode, refer to *_body_bias_domain* and PMC PDSLEEPCFG0 register descritpion in RM.

void POWER_EnableRunNBB(uint32_t mask)

Enable NBB mode for various domains in active mode. NBB can be a lower power option than RBB if the domain enters and exits low-power modes frequently since the wells don't charge and discharge as much on every entry/exit.

Parameters

- mask – : A bitmask of domains to enable NBB mode, refer to *_body_bias_domain* and PMC PDRUNCFG0 register descritpion in RM.

void POWER_EnableSleepNBB(uint32_t mask)

Enable NBB mode for various domains in deep sleep mode. NBB can be a lower power option than RBB if the domain enters and exits low-power modes frequently since the wells don't charge and discharge as much on every entry/exit.

Parameters

- mask – : A bitmask of domains to enable NBB mode, refer to *_body_bias_domain* and PMC PDSLEEPCFG0 register descritpion in RM.

uint32_t POWER_CalcVoltLevel(*power_regulator_t* regulator, uint32_t maxFreqHz, uint32_t miniVoltUV)

Get the minimum operative(active) voltage for VDD1 or VDD2 with the given maximum main clock frequency in the domain.

Parameters

- maxFreqHz – the maxiumum main clock frequency in Hz will be used for the domain.
- miniVoltUV – the minimum voltage in microvolt(uV) for VDD1 or VDD2. Should <= 1100000uV, 0 means use the main clock frequency to calculate voltage.

Returns

the calculated operative voltage in uV. 0xFFFFFFFFU means exceeded max supported value.

status_t POWER_ConfigRegulatorSetpointsForFreq(*power_regulator_t* regulator, uint32_t *maxFreqHz, uint32_t *miniVoltUV, uint32_t startPoint, uint32_t num)

Configure the setpoint operation for on-chip regulators(LDO1/LDO2) and LVD. This function can configure both the target output voltages and LVD to suitable values per the desired frequency for the specific setpoints. The setpoint that is actually selected is controlled by the aggregated value of PDRUNCFG[xxx_VSEL] and PDSLEEPCFG0[xxx_VSEL]. Use the *POWER_SelectRunSetpoint()* or *POWER_SelectSleepSetpoint()* to configure the setpoint selections. The voltage selected for each regulator must also be in ascending order (VSEL0 <= VSEL1 <= VSEL2 <= VSEL3).

Parameters

- maxFreqHz – the maxiumum main clock frequency in Hz will be used for the domain. 0 means use the miniVolt defined value and can be used to configure setpoint for DeepSleep.
- miniVolt – the minimum voltage in microvolt(uV) for VDD1 or VDD2. Should <= 1100000uV, 0 means use the main clock frequency to calculate voltage.
- startPoint – the start setpoint to be configured.
- num – the count of setpoints to be configured, the startPoint+num should not larger than 4.

Returns

kStatus_Success for successful configuration, kStatus_InvalidArgument for wrong parameter.

status_t POWER_ConfigRegulatorSetpoints(*power_regulator_t* regulator, const *power_regulator_voltage_t* *volt, const *power_lvd_voltage_t* *lvd)

Configure the setpoint operation for on-chip regulators and LVD. This function can configure both the target output voltages and LVD levels for each setpoint. The setpoint that is actually selected is controlled by the aggregated value of PDRUNCFG[xxx_VSEL] and PDSLEEPCFG0[xxx_VSEL]. Use the *POWER_SelectRunSetpoint()* or *POWER_SelectSleepSetpoint()* to configure the setpoint selections. The voltages selected for each regulator must also be in ascending order (VSEL0 <= VSEL1 <= VSEL2 <= VSEL3). The voltages selected for each LVD must also be in ascending order (LVL0 <= LVL1 <= LVL2 <= LVL3).

NOTE, Only valid regulator for sense domain is VDD1 LDO. Note, LDOVDD1VSEL and LVDVDD1CTRL are shared between the compute and sense domains. There is no aggregation to combine the values. Whichever domain wrote the register last determines the value. It is recommended that only one domain be used to configure VDD1 setpoints.

```
const power_regulator_voltage_t regulator = {
  .LDO.vsel0 = 700000,
  .LDO.vsel1 = 800000,
  .LDO.vsel2 = 900000,
  .LDO.vsel3 = 1000000,
};
const power_lvd_voltage_t lvd = {
  .VDD12.lvl0 = 600000,
  .VDD12.lvl1 = 700000,
  .VDD12.lvl2 = 800000,
  .VDD12.lvl3 = 900000,
};

ret = POWER_ConfigRegulatorSetpoints(kRegulator_Vdd1LDO, &regulator, &lvd);
```

Parameters

- regulator – which regulator or power domain to configure, refer to *power_regulator_t*.
- volt – regulator configuration pointer, refer to *power_regulator_voltage_t*. Note, only two setpoints are available for DCDC.
- lvd – LVD voltage configuration, refer to *power_lvd_voltage_t*.

Returns

kStatus_Success for succeed, kStatus_InvalidArgument for wrong arguments.

status_t POWER_ConfigLvdSetpoints(*power_regulator_t* regulator, const *power_lvd_voltage_t* *lvd)

Configure the setpoint operation for LVD. This function can configure LVD levels for each setpoint. The setpoint that is actually selected is controlled by the aggregated value of PDRUNCFG[xxx_VSEL] and PDSLEEPCFG0[xxx_VSEL]. Use the *POWER_SelectRunSetpoint()* or *POWER_SelectSleepSetpoint()* to configure the setpoint selections. The voltages selected for each LVD must be in ascending order (LVL0 <= LVL1 <= LVL2 <= LVL3).

NOTE, Only valid regulator for sense domain is VDD1 LDO. Note, LDOVDD1VSEL and LVDVDD1CTRL are shared between the compute and sense domains. There is no aggregation to combine the values. Whichever domain wrote the register last determines the value. It is recommended that only one domain be used to configure VDD1 setpoints.

```
const power_lvd_voltage_t lvd = {
  .VDD12.lvl0 = 600000,
```

(continues on next page)

(continued from previous page)

```
  .VDD12.lvl1 = 700000,
  .VDD12.lvl2 = 800000,
  .VDD12.lvl3 = 900000,
};

ret = POWER_ConfigLvdSetpoints(kRegulator_Vdd1LDO, &lvd);
```

Parameters

- regulator – which regulator or power domain to configure, refer to *power_regulator_t*.
- lvd – LVD voltage configuration, refer to *power_lvd_voltage_t*.

Returns

kStatus_Success for succeed, kStatus_InvalidArgument for wrong arguments.

void POWER_GetLvdSetpoints(*power_regulator_t* regulator, *power_lvd_voltage_t* *lvd)

Get the LVD setpoints configurations. This function can get the LVD levels for each setpoint. The setpoint that is actually selected is controlled by the aggregated value of PDRUNCFG[xxx_VSEL] and PDSLEEPCFG0[xxx_VSEL].

NOTE, Only valid regulator for sense domain is VDD1 LDO.

Parameters

- regulator – which regulator or power domain to get, refer to *power_regulator_t*.
- lvd – LVD voltage configuration, refer to *power_lvd_voltage_t*.

void POWER_SelectRunSetpoint(*power_regulator_t* regulator, uint32_t setpoint)

Select the setpoint(target voltage and LVD threshold) for active mode. The requests from the compute and sense domains are aggregated, so depending on the other domain's configuration the setpoint might not change or might change later (when the other domain also requests a setpoint change). Note, users should call *POWER_ApplyPD()* to make the change take effect.

Parameters

- regulator – which regulator or power domain to configure, refer to *power_regulator_t*.
- setpoint – select which setpoint is used. Valid options for DCDC are 0-1. Valid options for VDD1/VDD2 are 0-3.

void POWER_SelectSleepSetpoint(*power_regulator_t* regulator, uint32_t setpoint)

Select setpoint(target voltage and LVD threshold) in Deep Sleep (or lower) mode. Use *POWER_ConfigRegulatorSetpoints()* to change the target voltage and/or LVD threshold value corresponding to each setpoint. The requests from the compute and sense domains are aggregated,.

Parameters

- regulator – which regulator or power domain to configure, refer to *power_regulator_t*.
- setpoint – select which setpoint is used. Valid options for DCDC are 0-1. Valid options for VDD1/VDD2 are 0-3.

void POWER_SetRunRegulatorMode(*power_regulator_t* regulator, uint32_t mode)

Set the on-chip regulator mode in Run/Active mode. The requests from the compute and sense domains are aggregated.

Parameters

- regulator – which regulator or power domain to configure, refer to *power_regulator_t*.
- mode – set the workmode. Refer to *_power_dcdc_mode* for available DCDC modes, refer to *_power_ldo_mode* for LDO available modes.

void POWER_SetSleepRegulatorMode(*power_regulator_t* regulator, uint32_t mode)

Set the on-chip regulator mode in Deep Sleep mode. The requests from the compute and sense domains are aggregated.

Parameters

- regulator – which regulator or power domain to configure, refer to *power_regulator_t*.
- mode – set the workmode. Refer to *_power_dcdc_mode* for available DCDC modes, refer to *_power_ldo_mode* for LDO available modes.

void POWER_ResetIOBank(uint32_t mask)

Reset the IO bank. Sets the IO bank reset which tristates the ports. Intended to be used prior to powering off the VDDIO supplies.

Parameters

- mask – A bitmask of IO Banks to be reseted, power_io_bank_t and PMC PADCFG register. The register reset by cold reset.

void POWER_IOBankIsolationHold(uint32_t mask)

IO Bank Isolation Hold. Maintains IO bank isolation state after wake-up from FDSR/DPD modes. If clear, the IO bank's state will be retained during FDSR & DPD modes, but will be controllable after wake-up. If set, the IO bank will remain in retain mode until ISOCTRL cleared by software.

Parameters

- mask – A bitmask of IO Banks to be set, power_io_bank_t and PMC PADCFG register. The register reset by cold reset.

void POWER_IOBankClearIsolationHold(uint32_t mask)

Clear the IO bank Isolation Hold and regain state control. This bit cannot be cleared if associated domain is not powered.

Parameters

- mask – A bitmask of IO Banks to be clear, power_io_bank_t and PMC PADCFG register. The register reset by cold reset.

void POWER_EnterSleep(void)

Configures and enters in SLEEP low power mode.

void POWER_EnterDeepSleep(const uint32_t exclude_from_pd[7])

PMC Deep Sleep function call.

NOTE, the body bias and DCDC, LDO configurations in PMC_PDSLEECFG0 is not covered by this API, please use *POWER_EnableSleepRBB* or *POWER_EnableSleepNBB* and *POWER_SelectSleepSetpoint* for those settings.

NOTE, the MAINPLL and AUDIO PLL need special power up sequence, so it can't be power down/on automatically during the mode change. Application need turn off the PLL before the low power mode entry and re-initialize the PLL after wakeup.

Parameters

- exclude_from_pd – Bit mask of the SLEEPCON_SLEEPCFG and PMC_PDSLEEPCFG0 ~ PMC_PDSLEEPCFG5 that needs to be powered on during Deep Sleep mode. The configuration for PMC from Compute domain and Sense domain are aggregated, and the higher power mode always wins.

void POWER_RequestDSR(const uint32_t exclude_from_pd[7])

Sense domain requests entering Deep Sleep Retention mode API. Deep Sleep Retention (DSR) mode is a deeper low power state than Deep Sleep mode. DSR mode allows significant portions of the chip to be powered-down while maintaining some level of state retention. The Sense domain can enter DSR mode only when all other domains are in DSR mode. If the condition is false, the sense domain will be in a shallower power mode(Deep sleep mode).

NOTE, the body bias and DCDC, LDO configurations in PMC_PDSLEECFG0 is not covered by this API, please use *POWER_EnableSleepRBB* or *POWER_EnableSleepNBB* and *POWER_SelectSleepSetpoint* for those settings.

Parameters

- exclude_from_pd – Bit mask of the SLEEPCON_SLEEPCFG and PMC_PDSLEEPCFG0 ~ PMC_PDSLEEPCFG5 that needs to be powered on(bits cleared) during the power mode. The configuration for PMC from Compute domain and Sense domain are aggregated, and the higher power mode always wins.

void POWER_RequestDeepPowerDown(const uint32_t exclude_from_pd[7])

The domain requests to enter Deep Power Down mode. The Deep Power Down and Full Deep Power Down can occur only when both domains signal their intent to enter these lower power modes. The configuration for PMC from Compute domain and Sense domain are aggregated, and the higher power mode always wins.

Parameters

- exclude_from_pd – Bit mask of the SLEEPCON_SLEEPCFG and PMC_PDSLEEPCFG0 ~ PMC_PDSLEEPCFG5 that needs to be powered on during Deep Power Down.

void POWER_RequestFullDeepPowerDown(const uint32_t exclude_from_pd[7])

The domain requests to enter Full Deep Power Down mode. The Deep Power Down and Full Deep Power Down can occur only when both domains signal their intent to enter these lower power modes. The configuration for PMC from Compute domain and Sense domain are aggregated, and the higher power mode always wins.

Parameters

- exclude_from_pd – Bit mask of the SLEEPCON_SLEEPCFG and PMC_PDSLEEPCFG0 ~ PMC_PDSLEEPCFG5 that needs to be powered on during Deep Power Down.

void POWER_EnterPowerMode(*power_mode_cfg_t* mode, const uint32_t exclude_from_pd[7])

Power Library API to request entering different power mode. This API is used for requesting entering the target mode, the final SOC power mode depends on hardware aggregation. Note, the Sense domain can enter DSR mode only when Compute domain is in DSR. The SOC enters DPD or FDPD when both domain requested entering DPD or FDPD.

Parameters

- mode – Power mode to enter.
- exclude_from_pd – Bit mask of the SLEEPCON_SLEEPCFG and PMC_PDSLEEPCFG0 ~ PMC_PDSLEEPCFG5 that needs to be powered on during power mode selected.

uint32_t POWER_GetLibVersion(void)

Power Library API to return the library version.

Returns

version number of the power library

FSL_POWER_DRIVER_VERSION

power driver version 2.5.1.

POWER_RETRY_TIMES

Retry times for waiting flag.

POWER_DEFAULT_PMICMODE_DS

Define the default PMIC modes for power modes.

POWER_DEFAULT_PMICMODE_DSR

POWER_DEFAULT_PMICMODE_DPD

POWER_DEFAULT_PMICMODE_FDPD

POWER_DEFAULT_LVD_VOLT

Define the default LVD threshold.

POWER_LDO_SAFE_MARGIN(x)

Define the safe margin for regulator output voltage levels.

MAKE_PD_BITS(reg, slot)

GET_PD_REG_FROM_BITS(bit)

SLEEPCON_RCFG

PMC_PDRCFG0

PMC_PDRCFG1

PMC_PDRCFG2

PMC_PDRCFG3

PMC_PDRCFG4

PMC_PDRCFG5

SHA_MED_CCTRL0_OFFSET

SHA_MED_TCTRL0_OFFSET

SHA_SEN_TCTRL0_OFFSET

PRIVATE_CCTRL0_OFFSET

PRIVATE_TCTRL0_OFFSET

PRIVATE_TCTRL1_OFFSET

PRIVATE_TCTRL2_OFFSET

SHA_MED_CSTAT0_OFFSET

SHA_MEDSEN_TSTAT0_OFFSET

PRIVATE_CSTAT0_OFFSET

PRIVATE_TSTAT0_OFFSET

PRIVATE_TSTAT1_OFFSET

POWER_LP_REQ_CTRL_REG_OFFSET

POWER_LP_REQ_STATE_REG_OFFSET

POWER_LP_REQ_STATE_BIT_OFFSET

POWER_MAKE_LP_BITS(ctrl_reg, ctrl_bit, state_reg, state_bit)

GET_LP_CTRL_REG_FROM_BITS(bit)

GET_LP_CTRL_BIT_FROM_BITS(bit)

GET_LP_STATE_REG_FROM_BITS(bit)

GET_LP_STATE_BIT_FROM_BITS(bit)

uint32_t vsel0
: DCDC VSEL0. Voltage = 0.5 V + 6.25 mV * value of vsel0. Maximum value = 1150000uV.

uint32_t vsel1
: DCDC VSEL1. Voltage = 0.5 V + 6.25 mV * value of vsel1. Maximum value = 1150000uV.

struct *_power_regulator_voltage* DCDC

uint32_t vsel0
: LDO VSEL0. Voltage = 0.45 V + 12.5 mV * vsel0. Maximum value = 1150000uV.

uint32_t vsel1
: LDO VSEL1. Voltage = 0.45 V + 12.5 mV * vsel1. Maximum value = 1150000uV.

uint32_t vsel2
: LDO VSEL2. Voltage = 0.45 V + 12.5 mV * vsel2. Maximum value = 1150000uV.

uint32_t vsel3
: LDO VSEL3. Voltage = 0.45 V + 12.5 mV * vsel3. Maximum value = 1150000uV.

struct *_power_regulator_voltage* LDO

uint32_t vsel[4]

uint32_t lvl0
: VDDN LVD level0, uV.

uint32_t lvl1
: VDDN LVD level1, uV.

struct *_power_lvd_voltage* VDDN

uint32_t lvl0
: VDD1/2 LVD level0, uV.

uint32_t lvl1
: VDD1/2 LVD level1, uV.

uint32_t lvl2
: VDD1/2 LVD level2, uV.

uint32_t lvl3
: VDD1/2 LVD level3, uV.

struct *_power_lvd_voltage* VDD12

uint32_t lvl[4]

union __power__regulator__voltage
: *#include <fsl_power.h>* Regulator voltage configuraton. Configure the regulator voltage in uV.

union __power__lvd__voltage
: *#include <fsl_power.h>* LVD voltage configuraton. Configure the LVD voltage in uV. Falling trip = 0.5V + 10 mV * bitfield value.

struct DCDC

Public Members

uint32_t vsel0
DCDC VSEL0. Voltage = 0.5 V + 6.25 mV * value of vsel0. Maximum value = 1150000uV.

uint32_t vsel1
DCDC VSEL1. Voltage = 0.5 V + 6.25 mV * value of vsel1. Maximum value = 1150000uV.

struct LDO

Public Members

uint32_t vsel0
LDO VSEL0. Voltage = 0.45 V + 12.5 mV * vsel0. Maximum value = 1150000uV.

uint32_t vsel1
LDO VSEL1. Voltage = 0.45 V + 12.5 mV * vsel1. Maximum value = 1150000uV.

uint32_t vsel2
LDO VSEL2. Voltage = 0.45 V + 12.5 mV * vsel2. Maximum value = 1150000uV.

uint32_t vsel3
LDO VSEL3. Voltage = 0.45 V + 12.5 mV * vsel3. Maximum value = 1150000uV.

struct VDDN

Public Members

uint32_t lvl0
VDDN LVD level0, uV.

uint32_t lvl1
VDDN LVD level1, uV.

struct VDD12

Public Members

uint32_t lvl0
VDD1/2 LVD level0, uV.

uint32_t lvl1
VDD1/2 LVD level1, uV.

uint32_t lvl2
VDD1/2 LVD level2, uV.

uint32_t lvl3
VDD1/2 LVD level3, uV.

2.73 PUF: Physical Unclonable Function

2.74 Puf_v3_driver

FSL_PUF_V3_DRIVER_VERSION
PUFv3 driver version. Version 2.0.4.

Current version: 2.0.4

Change log:

- 2.0.4
  - Fix warning
- 2.0.3
  - Update for various PUF CTRL wrapper
- 2.0.2
  - Fix MISRA issue in driver.
- 2.0.1
  - Fix PUF initialization issue and update driver to reflect SoC header changes.
- 2.0.0
  - Initial version.

Values:

enumerator kStatus_PUF_OperationNotAllowed

enumerator kStatus_PUF_AcNotForThisProductPhase1

enumerator kStatus_PUF_AcNotForThisProductPhase2

enumerator kStatus_PUF_AcCorruptedPhase1

enumerator kStatus_PUF_AcCorruptedPhase2

enumerator kStatus_PUF_AcAuthFailedPhase1

enumerator kStatus_PUF_NBOOT_AcAuthFailedPhase2

enumerator kStatus_PUF_QualityVerificationFail

enumerator kStatus_PUF_ContextIncorrect

enumerator kStatus_PUF_DestinationNotAllowed

enumerator kStatus_PUF_Failure

typedef uint32_t puf_endianness_t

typedef uint32_t puf_key_dest_t

typedef uint32_t puf_key_scope_t

typedef uint32_t puf_result_code_t

typedef uint32_t puf_sec_level_t

puf_endianness_t dataEndianness

uint8_t CKGATING

puf_key_scope_t keyScopeStarted

puf_key_scope_t keyScopeEnrolled

uint32_t userCtx0

uint32_t userCtx1

void PUF_GetDefaultConfig(*puf_config_t* *conf)

brief Sets the default configuration of PUF

This function initialize PUF config structure to default values.

Parameters

- conf – PUF configuration structure

status_t PUF_Init(PUF_Type *base, *puf_config_t* *conf)

brief Initialize PUF

This function enables power to PUF block and waits until the block initializes.

Parameters

- conf – PUF configuration structure

Returns

Status of the init operation

void PUF_Deinit(PUF_Type *base, *puf_config_t* *conf)

brief Denitialize PUF

This function disables power to PUF SRAM and peripheral clock.

Parameters

- base – PUF peripheral base address
- conf – PUF configuration structure

status_t PUF_Enroll(PUF_Type *base, uint8_t *activationCode, size_t activationCodeSize, uint8_t *score)

brief Enroll PUF

This function derives a digital fingerprint, generates the corresponding Activation Code (AC) and returns it to be stored in an NVM or a file. This step needs to be performed only once for each device. This function may be permanently disallowed by a fuse.

Parameters

- base – PUF peripheral base address
- activationCode – **[out]** Word aligned address of the resulting activation code.
- activationCodeSize – Size of the activationCode buffer in bytes. Shall be FSL_FEATURE_PUF_ACTIVATION_CODE_SIZE bytes.
- score – Value of the PUF Score that was obtained during the enroll operation.

Returns

Status of enroll operation.

status_t PUF_Start(PUF_Type *base, const uint8_t *activationCode, size_t activationCodeSize, uint8_t *score)

brief Start PUF

The Activation Code generated during the Enroll operation is used to reconstruct the digital fingerprint. This needs to be done after every power-up and reset.

Parameters

- base – PUF peripheral base address
- activationCode – **[in]** Word aligned address of the input activation code.
- activationCodeSize – Size of the activationCode buffer in bytes. Shall be FSL_FEATURE_PUF_ACTIVATION_CODE_SIZE bytes.
- score – Value of the PUF Score that was obtained during the start operation. return Status of start operation.

status_t PUF_Stop(PUF_Type *base)

brief Stop PUF

The Stop operation removes all key material from PUF flipflops and PUF SRAM, and sets PUF to the Stopped state.

Parameters

- base – PUF peripheral base address

Returns

Status of stop operation.

status_t PUF_GetKey(PUF_Type *base, *puf_key_ctx_t* *keyCtx, *puf_key_dest_t* keyDest, uint8_t *key, size_t keySize)

brief PUF Get Key

The Get Key operation derives a key from the intrinsic PUF key and externally provided context.

Parameters

- base – PUF peripheral base address
- keyCtx – PUF key context struct
- keyDest – output destination of the derived PUF key
- key – **[out]** Word aligned address of output key (only used when kPUF_KeyDestRegister).
- keySize – Size of the derived key in bytes.

Returns
Status of get key operation.

status_t PUF_WrapGeneratedRandom(PUF_Type *base, *puf_key_ctx_t* *keyCtx, size_t keySize, uint8_t *keyCode, size_t keyCodeSize)

brief PUF Wrap generated random

The Wrap Generated Random operation wraps a random key into a Key Code (KC).

Parameters

- base – PUF peripheral base address
- keyCtx – PUF key context struct
- keySize – Size of the key to be generated in bytes.
- keyCode – **[out]** Word aligned address of the resulting key code.
- keyCodeSize – Size of the output keycode in bytes.

Returns
Status of wrap generated random operation.

status_t PUF_Wrap(PUF_Type *base, *puf_key_ctx_t* *keyCtx, uint8_t *userKey, size_t userKeySize, uint8_t *keyCode, size_t keyCodeSize)

brief PUF Wrap user key

The Wrap operation wraps a user defined key into a Key Code (KC).

Parameters

- base – PUF peripheral base address
- keyCtx – PUF key context struct.
- userKey – Word aligned address of input user key.
- userKeySize – Size of the key to be wrapped in bytes.
- keyCode – **[out]** Word aligned address of the resulting key code.
- keyCodeSize – Size of the output keycode in bytes.

Returns
Status of wrap operation.

status_t PUF_Unwrap(PUF_Type *base, *puf_key_dest_t* keyDest, uint8_t *keyCode, size_t keyCodeSize, uint8_t *key, size_t keySize)

brief PUF Unwrap user key

The unwrap operation unwraps the key from a previously created Key Code (KC)

Parameters

- base – PUF peripheral base address
- keyDest – output destination of the unwraped PUF key
- keyCode – **[in]** Word aligned address of the input key code.
- keyCodeSize – Size of the input keycode in bytes.

- key – Word aligned address of output key (only used when kPUF_KeyDestRegister).
- keySize – Size of the key to be generated in bytes.

Returns
Status of unwrap operation.

status_t PUF_GenerateRandom(PUF_Type *base, uint8_t *data, size_t size)

brief Generate Random

The Generate Random operation outputs the requested amount of random data as specified in a provided context.

Parameters

- base – PUF peripheral base address
- size – Size of random data to be genarated in bytes.

Returns
Status of generate random operation.

status_t PUF_Zeroize(PUF_Type *base)

brief Zeroize PUF

This function clears all PUF internal logic and puts the PUF to zeroized state.

Parameters

- base – PUF peripheral base address

Returns
Status of the zeroize operation.

status_t PUF_Test(PUF_Type *base, uint8_t *score)

brief Test PUF

With the Test PUF operation, diagnostics about the PUF quality is collected and presented in a PUF score.

Parameters

- base – PUF peripheral base address
- score – Value of the PUF Score that was obtained during the enroll operation.

Returns
Status of the test operation.

static inline void PUF_BlockCommand(PUF_Type *base, uint32_t mask)

Blocks specified PUF commands.

This function blocks PUF commands specified by mask parameter.

Parameters

- base – PUF peripheral base address
- mask – Mask of parameters which should be blocked until power-cycle.

Returns
Status of the test operation.

status_t PUF_SetLock(PUF_Type *base, *puf_sec_level_t* securityLevel)

brief Set lock of PUF operation

Lock the security level of PUF block until key generate, wrap or unwrap operation is completed. Note: Only security level defined in SEC_LOCK register can use PUFv3 or change its security level. Default setting after leaving ROM is Secure-Privilege

Parameters

- base – PUF peripheral base address
- securityLevel – Security level of PUF block.

Returns
Status of the test operation.

status_t PUF_SetCtxMask(PUF_Type *base, uint32_t appCtxMask)

brief Set App Context mask

This function sets Application defined context mask used in conjunction with key user context 2. Whenever bit in this register is 1, corresponding bit in user context 2 provided during key code creation should be zero only.

This register is only modifiable by task running at secure-privilege level.

Parameters

- base – PUF peripheral base address
- appCtxMask – Value of the Application defined context mask.

Returns
Status of the test operation.

kPUF_EndianLittle

kPUF_EndianBig

kPUF_KeyDestRegister

kPUF_KeyDestKeyBus

kPUF_KeyDestInvalid

kPUF_KeyAllowRegister

kPUF_KeyAllowKeyBus

kPUF_KeyAllowAll

kPUF_ResultOK

kPUF_AcNotForThisProductPhase1

kPUF_AcNotForThisProductPhase2

kPUF_AcCorruptedPhase1

kPUF_AcCorruptedPhase2

kPUF_AcAuthFailedPhase1

kPUF_AcAuthFailedPhase2

kPUF_QualityVerificationFail

kPUF_ContextIncorrect

kPUF_DestinationNotAllowed

kPUF_Failure

kPUF_NonsecureUser

kPUF_NonsecurePrivilege

kPUF_SecureUser

kPUF_SecurePrivilege

PUF_ACTIVATION_CODE_SIZE

PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(x)

SEC_LOCK_PATTERN

struct puf_config_t

#include <fsl_puf_v3.h>

struct puf_key_ctx_t

#include <fsl_puf_v3.h>

2.75 Pvts

enum pvts_sensor

PVTS sensors, each PVTS provides 2 sets of sensors to monitor 2 CPUs in same power domain.

Values:

enumerator kPVTS_Sensor0

Sensor 0 for ARM core.

enumerator kPVTS_Sensor1

Sensor 1 for HiFi core.

typedef enum *pvts_sensor* pvts_sensor_t

PVTS sensors, each PVTS provides 2 sets of sensors to monitor 2 CPUs in same power domain.

typedef uint8_t pvts_delay_t

PVTS delay type.

void PVTS_Init(void)

Initializes the PVTS peripheral.

```
uint32_t delay;
PVTS_ReadDelayFromOTP(false, kPVTS_Vdd2Com, CLOCK_GetFreq(kCLOCK_CoreSysClk), &
↪delay);
PVTS_Init();
EnableIRQ((IRQn_Type) PVTS0_IRQn);
PVTS_EnableAlertCount(kPVTS_Sensor0);
PVTS_SetDelay(kPVTS_Sensor0, PVTS_GET_DELAY0_FROM_FUSE_VALUE(delay));
PVTS_Start(kPVTS_Sensor0);
```

Note: The complete initialization should look like below.

void PVTS_Deinit(void)

De-initializes the PVTS peripheral.

void PVTS_Start(*pvts_sensor_t* sensor)

Enables PVTS sensing.

Parameters

- sensor – the sensor to start.

void PVTS_Stop(*pvts_sensor_t* sensor)

Disables PVTS sensing.

Parameters

- sensor – the sensor to stop.

status_t PVTS_GetDelay(*pvts_sensor_t* sensor, *pvts_delay_t* *delay_value)

Reads the current PVTS delay value.

Parameters

- sensor – the sensor to read.
- delay_value – Pointer to a pvts_delay_t where the PVT delay value will be placed.

Return values

- kStatus_Success – Successfully read the delay.
- kStatus_Fail – Failed to read delay because it has not been set.

status_t PVTS_SetDelay(*pvts_sensor_t* sensor, *pvts_delay_t* delay)

Sets the delay value for the PVTS.

Note: Must call *PVTS_Start()* to start sensing.

Parameters

- sensor – the sensor to configure.
- delay – value read using PVTS_ReadDelayFromOTP().

Return values

- kStatus_Success – Successfully sets the delay.
- kStatus_InvalidArgument – Failed to set delay. Invalid parameter passed in.

void PVTS_EnableAlertCount(*pvts_sensor_t* sensor)

Enables the 20-bit PVT alert counter.

Enables counting the amount of times the PVTS interrupt has triggered.

Note: It is up to the user to clear the counter.

Parameters

- sensor – the sensor to configure.

void PVTS_DisableAlertCount(*pvts_sensor_t* sensor)

Disables the 20-bit PVTS alert counter.

Parameters

- sensor – the sensor to set.

uint32_t PVTS_GetAlertCount(*pvts_sensor_t* sensor)

Reads the 20-bit PVTS alert counter.

Parameters

- sensor – the sensor to read.

Returns

20-bit PVT alert counter value.

void PVTS_ClearAlertCount(*pvts_sensor_t* sensor)

Resets the 20-bit PVTS alert counter.

Parameters

- sensor – the sensor to clear.

FSL_PVTS_DRIVER_VERSION

pvts driver version 2.1.0.

PVTS_GET_DELAY0_FROM_FUSE_VALUE(fuse)

Calculate delay0 value from fuse word.

PVTS_GET_DELAY1_FROM_FUSE_VALUE(fuse)

Calculate delay1 value from fuse word.

PVTS_GET_A_PARAMETER_FROM_FUSE_VALUE(fuse)

Calculate A parameter value from fuse word.

PVTS_GET_B_PARAMETER_FROM_FUSE_VALUE(fuse)

Calculate B parameter value from fuse word.

PVTS_DELAY_MAX

Valide max PVTS delay

PVTS_DELAY_MIN

Valide min PVTS delay

2.76 Reset Driver

enum _rstctl_reset_source

Enumeration for system reset status bits.

Defines the enumeration for system reset status bits in SYSRSTSTAT register

Values:

enumerator kRSTCTL_SourceVddPor
VDD Power-On Reset(POR)

enumerator kRSTCTL_SourcePad
PPAD Reset

enumerator kRSTCTL_SourceIspAp
ISP_AP reset Reset

enumerator kRSTCTL_SourceItrcSw
ITRC_SW (Intrusion and Tamper Response Controller SW) Reset

enumerator kRSTCTL_SourceCpu0
VDD2_COMP Core Reset

enumerator kRSTCTL_SourceCpu1
VDD1_SENSE Core Reset

enumerator kRSTCTL_SourceWwdt0
WatchDog Timer 0 Reset

enumerator kRSTCTL_SourceWwdt1
WatchDog Timer 1 Reset

enumerator kRSTCTL_SourceWwdt2
WatchDog Timer 2 Reset

enumerator kRSTCTL_SourceWwdt3
WatchDog Timer 3 Reset

enumerator kRSTCTL_SourceCdog0
Code WatchDog Timer 0 Reset

enumerator kRSTCTL_SourceCdog1
Code WatchDog Timer 1 Reset

enumerator kRSTCTL_SourceCdog2
Code WatchDog Timer 2 Reset

enumerator kRSTCTL_SourceCdog3
Code WatchDog Timer 3 Reset

enumerator kRSTCTL_SourceCdog4
Code WatchDog Timer 4 Reset

enumerator kRSTCTL_SourceAll

enum _RSTCTL_RSTn
Enumeration for peripheral reset control bits.
Defines the enumeration for peripheral reset control bits in RSTCLTx registers
Values:

enumerator kIOPCTL0_RST_SHIFT_RSTn
IOPCTL0 reset control

enumerator kELS_RST_SHIFT_RSTn
ELS S50 reset control

enumerator kDMA0_RST_SHIFT_RSTn
eDMA0 reset control

enumerator kDMA1_RST_SHIFT_RSTn
eDMA1 reset control

enumerator kPKC_RST_SHIFT_RSTn
PKC reset control

enumerator kXSPI0_RST_SHIFT_RSTn
XSPI0 reset control

enumerator kXSPI1_RST_SHIFT_RSTn
XSPI1 reset control

enumerator kGPIO0_RST_SHIFT_RSTn
GPIO0 reset control

enumerator kGPIO1_RST_SHIFT_RSTn
GPIO1 reset control

enumerator kGPIO2_RST_SHIFT_RSTn
GPIO2 reset control

enumerator kGPIO3_RST_SHIFT_RSTn
GPIO3 reset control

enumerator kGPIO4_RST_SHIFT_RSTn
GPIO4 reset control

enumerator kGPIO5_RST_SHIFT_RSTn
GPIO5 reset control

enumerator kGPIO6_RST_SHIFT_RSTn
GPIO6 reset control

enumerator kGPIO7_RST_SHIFT_RSTn
GPIO7 reset control

enumerator kSCT0_RST_SHIFT_RSTn
SCT0 reset control

enumerator kFC0_RST_SHIFT_RSTn
LP_FLEXCOMM0 reset control

enumerator kFC1_RST_SHIFT_RSTn
LP_FLEXCOMM1 reset control

enumerator kFC2_RST_SHIFT_RSTn
LP_FLEXCOMM2 reset control

enumerator kFC3_RST_SHIFT_RSTn
LP_FLEXCOMM3 reset control

enumerator kFC4_RST_SHIFT_RSTn
LP_FLEXCOMM4 reset control

enumerator kFC5_RST_SHIFT_RSTn
LP_FLEXCOMM5 reset control

enumerator kFC6_RST_SHIFT_RSTn
LP_FLEXCOMM6 reset control

enumerator kFC7_RST_SHIFT_RSTn
LP_FLEXCOMM7 reset control

enumerator kFC8_RST_SHIFT_RSTn
LP_FLEXCOMM8 reset control

enumerator kFC9_RST_SHIFT_RSTn
LP_FLEXCOMM9 reset control

enumerator kFC10_RST_SHIFT_RSTn
LP_FLEXCOMM10 reset control

enumerator kFC11_RST_SHIFT_RSTn
LP_FLEXCOMM11 reset control

enumerator kFC12_RST_SHIFT_RSTn
LP_FLEXCOMM12 reset control

enumerator kFC13_RST_SHIFT_RSTn
LP_FLEXCOMM13 reset control

enumerator kSAI0_RST_SHIFT_RSTn
SAI0 reset control

enumerator kSAI1_RST_SHIFT_RSTn
SAI1 reset control

enumerator kSAI2_RST_SHIFT_RSTn
SAI2 reset control

enumerator kI3C0_RST_SHIFT_RSTn
I3C0 reset control

enumerator kI3C1_RST_SHIFT_RSTn
I3C1 reset control

enumerator kCRC0_RST_SHIFT_RSTn
CRC0 reset control

enumerator kCTIMER0_RST_SHIFT_RSTn
CTIMER0 reset control

enumerator kCTIMER1_RST_SHIFT_RSTn
CTIMER1 reset control

enumerator kCTIMER2_RST_SHIFT_RSTn
CTIMER2 reset control

enumerator kCTIMER3_RST_SHIFT_RSTn
CTIMER3 reset control

enumerator kCTIMER4_RST_SHIFT_RSTn
CTIMER4 reset control

enumerator kMRT0_RST_SHIFT_RSTn
MRT0 reset control

enumerator kUTICK0_RST_SHIFT_RSTn
UTICK0 reset control

enumerator kSEMA424_RST_SHIFT_RSTn
SEMA42_4 reset control

enumerator kMU4_RST_SHIFT_RSTn
MU4 reset control

enumerator kSYSPM2_RST_SHIFT_RSTn
SYSPM_XSPI0 reset control

enumerator kSYSPM3_RST_SHIFT_RSTn
SYSPM_XSPI1 reset control

enumerator kSAFO_SGI_RST_SHIFT_RSTn
SAFO_SGI reset control

enumerator kINPUTMUX0_RST_SHIFT_RSTn
INPUTMUX0 reset control

enumerator kFREQME_RST_SHIFT_RSTn
FREQME0 reset control

enumerator kSYSPM0_RST_SHIFT_RSTn
SYSPM_PC reset control

enumerator kSYSPM1_RST_SHIFT_RSTn
SYSPM_PS reset control

enumerator kNPU0_RST_SHIFT_RSTn
NPU0 reset control

enumerator kHIFI4_RST_SHIFT_RSTn
HiFi4 reset control

enumerator kHIFI4_DEBUG_RST_SHIFT_RSTn
HiFi4 Debug reset control

enumerator kHIFI1_RST_SHIFT_RSTn
HiFi1 reset control

enumerator kHIFI1_DEBUG_RST_SHIFT_RSTn
HiFi1 Debug reset control

enumerator kDMA2_RST_SHIFT_RSTn
eDMA2 reset control

enumerator kDMA3_RST_SHIFT_RSTn
eDMA3 reset control

enumerator kFC17_RST_SHIFT_RSTn
LP_FLEXCOMM17 reset control

enumerator kFC18_RST_SHIFT_RSTn
LP_FLEXCOMM18 reset control

enumerator kFC19_RST_SHIFT_RSTn
LP_FLEXCOMM19 reset control

enumerator kFC20_RST_SHIFT_RSTn
LP_FLEXCOMM20 reset control

enumerator kSAI3_RST_SHIFT_RSTn
SAI3 reset control

enumerator kI3C2_RST_SHIFT_RSTn
I3C2 reset control

enumerator kI3C3_RST_SHIFT_RSTn
I3C3 reset control

enumerator kGPIO8_RST_SHIFT_RSTn
GPIO8 reset control

enumerator kGPIO9_RST_SHIFT_RSTn
GPIO9 reset control

enumerator kGPIO10_RST_SHIFT_RSTn
GPIO10 reset control

enumerator kCTIMER5_RST_SHIFT_RSTn
CTIMER5 reset control

enumerator kCTIMER6_RST_SHIFT_RSTn
CTIMER6 reset control

enumerator kCTIMER7_RST_SHIFT_RSTn
CTIMER7 reset control

enumerator kMRT1_RST_SHIFT_RSTn
MRT1 reset control

enumerator kUTICK1_RST_SHIFT_RSTn
UTICK1 reset control

enumerator kMU3_RST_SHIFT_RSTn
MU3 reset control

enumerator kSEMA423_RST_SHIFT_RSTn
SEMA42_3 reset control

enumerator kPVTS1_RST_SHIFT_RSTn
PVTS1 reset control

enumerator kIOPCTL2_RST_SHIFT_RSTn
IOPCTL2 reset control

enumerator kIOPCTL1_RST_SHIFT_RSTn
IOPCTL1 reset control

enumerator kCPU1_RST_SHIFT_RSTn
CPU1 reset control

enumerator kMU0_RST_SHIFT_RSTn
MU0 reset control

enumerator kMU1_RST_SHIFT_RSTn
MU1 reset control

enumerator kMU2_RST_SHIFT_RSTn
MU2 reset control

enumerator kSEMA420_RST_SHIFT_RSTn
SEMA42_0 reset control

enumerator kADC0_RST_SHIFT_RSTn
ADC0 reset control

enumerator kSDADC0_RST_SHIFT_RSTn
SDADC0 reset control

enumerator kACMP0_RST_SHIFT_RSTn
ACMP0 reset control

enumerator kPDM_RST_SHIFT_RSTn
MICFIL/PDM reset control

enumerator kINPUTMUX1_RST_SHIFT_RSTn
INPUTMUX1 reset control

enumerator kLPI2C15_RST_SHIFT_RSTn
LPI2C15 reset control

enumerator kVGPU_RST_SHIFT_RSTn
VGPU reset control

enumerator kLCDIF_RST_SHIFT_RSTn
LCDIF reset control

enumerator kMIPI_DSI_CTRL_RST_SHIFT_RSTn
MIPIDSI reset control

enumerator kEZHV_RST_SHIFT_RSTn
EZHV reset control

enumerator kJPEGDEC_RST_SHIFT_RSTn
JPEGDEC reset control

enumerator kPNGDEC_RST_SHIFT_RSTn
PNGDEC reset control

enumerator kXSPI2_RST_SHIFT_RSTn
XSPI2 and MMU2 reset control

enumerator kLPSPI14_RST_SHIFT_RSTn
LPSPI14 reset control

enumerator kLPSPI16_RST_SHIFT_RSTn
LPSPI16 reset control

enumerator kFLEXIO0_RST_SHIFT_RSTn
FLEXIO0 reset control

enumerator kUSB0_RST_SHIFT_RSTn
USB0 reset control

enumerator kUSBPHY0_RST_SHIFT_RSTn
USBPHY0 reset control

enumerator kUSB1_RST_SHIFT_RSTn
USB1 reset control

enumerator kUSDHC0_RST_SHIFT_RSTn
USDHC0 reset control

enumerator kUSDHC1_RST_SHIFT_RSTn
USDHC1 reset control

Flash device current running mode.
Values:

enumerator kFlashInstMode_ExtendedSpi

enumerator kFlashInstMode_0_4_4_SDR

enumerator kFlashInstMode_0_4_4_DDR

enumerator kFlashInstMode_DPI_SDR

enumerator kFlashInstMode_DPI_DDR

enumerator kFlashInstMode_QPI_SDR

enumerator kFlashInstMode_QPI_DDR

enumerator kFlashInstMode_OPI_SDR

enumerator kFlashInstMode_OPI_DDR

flash reset sequence of standard SPI mode.

Values:

enumerator kRestoreSeqence_None

enumerator kRestoreSeqence_QPI_4_0xFFs

enumerator kRestoreSeqence_QPI_Mode_0x00

enumerator kRestoreSeqence_Send_F0

enumerator kRestoreSeqence_Send_66_99

enumerator kRestoreSeqence_Send_6699_9966

enumerator kRestoreSeqence_Send_06_FF

enumerator kRestoreSeqence_QPI_5_0xFFs

enumerator kRestoreSeqence_Send_QPI_8_0xFFs

enumerator kRestoreSeqence_Wakep_0xAB

enumerator kRestoreSeqence_Wakep_0xAB_54

typedef enum *_rstctl_reset_source* rstctl_reset_source_t

Enumeration for system reset status bits.

Defines the enumeration for system reset status bits in SYSRSTSTAT register

typedef enum *_RSTCTL_RSTn* RSTCTL_RSTn_t

Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in RSTCLTx registers

typedef union *_FLASH_run_context_t* FLASH_run_context_t

Flash context.

Defines the flash context parameter used by ROM.

typedef *RSTCTL_RSTn_t* reset_ip_name_t

IP reset handle.

void RESET_SetPeripheralReset(*reset_ip_name_t* peripheral)

Assert reset to peripheral.

Asserts reset signal to specified peripheral module.

Parameters

- peripheral – Assert reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET_ClearPeripheralReset(*reset_ip_name_t* peripheral)

Clear reset to peripheral.

Clears reset signal to specified peripheral module, allows it to operate.

Parameters

- peripheral – Clear reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

void RESET_PeripheralReset(*reset_ip_name_t* peripheral)

Reset peripheral module.

Reset peripheral module.

Parameters

- peripheral – Peripheral to reset. The enum argument contains encoding of reset register and reset bit position in the reset register.

static inline void RESET_ReleasePeripheralReset(*reset_ip_name_t* peripheral)

Release peripheral module.

Release peripheral module.

Parameters

- peripheral – Peripheral to release. The enum argument contains encoding of reset register and reset bit position in the reset register.

static inline uint32_t RESET_GetPreviousResetSources(void)

Gets the reset source status which caused a previous reset.

This function gets the status of the latest reset event. Use source masks defined in the rstctl_reset_source_t to get the desired source status.

Returns

All reset source status bit map.

static inline void RESET_ClearResetSources(uint32_t sourceMasks)

Clears the reset source status.

This function clears the system reset flags indicated by source masks.

Parameters

- sourceMasks – reset source status bit map

static inline uint32_t RESET_GetDomainResetStatus(void)

Gets the reset status of domains.

This function gets the reset status of domains.

Returns

All reset source status bit map.

FSL_RESET_DRIVER_VERSION

reset driver version 2.1.0.

RST_CTL0_PSCCTL0

Reset control registers index.

RST_CTL0_PSCCTL1

RST_CTL0_PSCCTL2

RST_CTL0_PSCCTL3

RST_CTL0_PSCCTL4

RST_CTL0_PSCCTL5

RST_CTL1_PSCCTL0

RST_CTL2_PSCCTL0

RST_CTL3_PSCCTL0

RST_CTL3_PSCCTL1

RST_CTL4_PSCCTL0

RST_CTL4_PSCCTL1

ADC_RSTS

Array initializers with peripheral reset bits

SDADC_RSTS

CRC_RSTS

CTIMER_RSTS

MIPI_DSI_RSTS

LCDIF_RSTS

LP_FLEXCOMM_RSTS

FLEXIO_RSTS

FREQME_RSTS_N

XSPI_RSTS

GPIO_RSTS

I3C_RSTS

MRT_RSTS

PINT_RSTS

PNGDEC_RSTS

JPEGDEC_RSTS

SCT_RSTS

SEMA42_RSTS

SAI_RSTS

USDHC_RSTS

UTICK_RSTS

uint8_t reseverd0
: Reserved.

uint8_t current_mode
: Flash device current running mode.

uint8_t reserved1
: Reserved.

uint8_t restore_sequence
: Flash reset sequence of standard SPI mode.

struct *_FLASH_run_context_t* B

uint32_t U

union __FLASH__run__context__t

#include <fsl_reset.h> Flash context.

Defines the flash context parameter used by ROM.

struct B

Public Members

uint8_t reseverd0

Reserved.

uint8_t current__mode

Flash device current running mode.

uint8_t reserved1

Reserved.

uint8_t restore__sequence

Flash reset sequence of standard SPI mode.

2.77 Runbootloader

typedef struct *_user_app_boot_invoke_option* user__app__boot__invoke__option__t

void bootloader__user__entry(void *arg)

Run the Bootloader API to force into the ISP mode base on the user arg.

Parameters

- arg – Indicates API prototype fields definition. Refer to the above user_app_boot_invoke_option_t structure

uint32_t bootloader__version(void)

Get Bootloader version.

Return values

Bootloader – version.

const char *bootloader__copyright(void)

Get Bootloader Copyright.

Return values

Pointer – to Bootloader copyright header

uint32_t reserved0

uint32_t recovery__boot__cfg0

uint32_t recovery__boot__cfg1

uint32_t reserved1

uint32_t instance

uint32_t boot_interface

uint32_t mode

uint32_t tag

struct *_user_app_boot_invoke_option* B

uint32_t U

union *_user_app_boot_invoke_option* option

struct __user__app__boot__invoke__option
#include <fsl_romapi.h>

union option

Public Members

struct *_user_app_boot_invoke_option* B

uint32_t U

struct B

2.78 SAI: Serial Audio Interface

2.79 SAI Driver

void SAI_Init(I2S_Type *base)
Initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_Init is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_Deinit(I2S_Type *base)
De-initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_TxReset(I2S_Type *base)

Resets the SAI Tx.

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_RxReset(I2S_Type *base)

Resets the SAI Rx.

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_TxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Tx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Tx, false means disable.

void SAI_RxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Rx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Rx, false means disable.

static inline void SAI_TxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_TxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Tx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

void SAI_TxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Transmitter bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_RxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Receiver bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_TxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Transmitter Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_RxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Receiver Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_SetMasterClockConfig(I2S_Type *base, *sai_master_clock_t* *config)

Master clock configurations.

Parameters

- base – SAI base pointer.
- config – master clock configurations.

void SAI_TxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI transmitter fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_RxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI receiver fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_TxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI transmitter Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_RxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI receiver Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_TxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI transmitter Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_RxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI receiver Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_TxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI transmitter configurations.

Parameters

- base – SAI base pointer.
- config – transmitter configurations.

void SAI_RxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI receiver configurations.

Parameters

- base – SAI base pointer.
- config – receiver configurations.

void SAI_GetClassicI2SConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get classic I2S mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetLeftJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get left justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetRightJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get right justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetTDMConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, uint32_t dataWordNum, uint32_t saiChannelMask)

Get TDM mode configurations.

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.

- bitWidth – audio data word width.
- dataWordNum – word number in one frame.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetDSPConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get DSP mode configurations.

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
SAI_TxSetConfig(base, config)
```

Note: DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
config->frameSync.frameSyncEarly    = true;
SAI_TxSetConfig(base, config)
```

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to enable.

static inline uint32_t SAI_TxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Tx status flag value. Use the Status Mask to get the status value needed.

static inline void SAI_TxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Tx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following source if defined:
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

static inline uint32_t SAI_RxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

static inline void SAI_RxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Rx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following sources if defined.
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

void SAI_TxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_RxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_TxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Tx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_RxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Rx channel FIFO enable mask.

Parameters

- base – SAI base pointer

- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_TxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Tx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_RxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Rx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_TxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Tx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

void SAI_RxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Rx FIFO packing feature.

Parameters

- base – SAI base pointer.

- pack – FIFO pack type. It is element of sai_fifo_packing_t.

static inline void SAI_TxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Tx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_RxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Rx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_TxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Tx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline void SAI_RxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Rx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be a combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uintptr_t SAI_TxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Tx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

static inline uintptr_t SAI_RxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Rx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

void SAI_WriteBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

void SAI_WriteMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data to multi channel using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

static inline void SAI_WriteData(I2S_Type *base, uint32_t channel, uint32_t data)

Writes data into SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- data – Data needs to be written.

void SAI_ReadBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

void SAI_ReadMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives multi channel data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

static inline uint32_t SAI_ReadData(I2S_Type *base, uint32_t channel)

Reads data from the SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.

Returns

Data in SAI FIFO.

void SAI_TransferTxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Tx handle.

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

void SAI_TransferRxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Rx handle.

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

void SAI_TransferTxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI transmitter transfer configurations.

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – tranmitter configurations.

void SAI_TransferRxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI receiver transfer configurations.

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – receiver configurations.

status_t SAI_TransferSendNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_TxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferReceiveNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferGetSendCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a set byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count sent.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

status_t SAI_TransferGetReceiveCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a received byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count received.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void SAI_TransferAbortSend(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current send.

Note: This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferAbortReceive(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferTerminateSend(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceive(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_TransferRxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_DriverIRQHandler(uint32_t instance)

SAI driver IRQ handler common entry.

This function provides the common IRQ request entry for SAI.

Parameters

- instance – SAI instance.

FSL_SAI_DRIVER_VERSION

Version 2.4.11

_sai_status_t, SAI return status.

Values:

enumerator kStatus_SAI_TxBusy

SAI Tx is busy.

enumerator kStatus_SAI_RxBusy

SAI Rx is busy.

enumerator kStatus_SAI_TxError

SAI Tx FIFO error.

enumerator kStatus_SAI_RxError

SAI Rx FIFO error.

enumerator kStatus_SAI_QueueFull

SAI transfer queue is full.

enumerator kStatus_SAI_TxIdle

SAI Tx is idle

enumerator kStatus_SAI_RxIdle
SAI Rx is idle

_sai_channel_mask,.sai channel mask value, actual channel numbers is depend soc specific
Values:

enumerator kSAI_Channel0Mask
channel 0 mask value

enumerator kSAI_Channel1Mask
channel 1 mask value

enumerator kSAI_Channel2Mask
channel 2 mask value

enumerator kSAI_Channel3Mask
channel 3 mask value

enumerator kSAI_Channel4Mask
channel 4 mask value

enumerator kSAI_Channel5Mask
channel 5 mask value

enumerator kSAI_Channel6Mask
channel 6 mask value

enumerator kSAI_Channel7Mask
channel 7 mask value

enum _sai_protocol
Define the SAI bus type.
Values:

enumerator kSAI_BusLeftJustified
Uses left justified format.

enumerator kSAI_BusRightJustified
Uses right justified format.

enumerator kSAI_BusI2S
Uses I2S format.

enumerator kSAI_BusPCMA
Uses I2S PCM A format.

enumerator kSAI_BusPCMB
Uses I2S PCM B format.

enum _sai_master_slave
Master or slave mode.
Values:

enumerator kSAI_Master
Master mode include bclk and frame sync

enumerator kSAI_Slave
Slave mode include bclk and frame sync

enumerator kSAI_Bclk_Master_FrameSync_Slave
bclk in master mode, frame sync in slave mode

enumerator kSAI_Bclk_Slave_FrameSync_Master
bclk in slave mode, frame sync in master mode

enum _sai_mono_stereo
Mono or stereo audio format.
Values:

enumerator kSAI_Stereo
Stereo sound.

enumerator kSAI_MonoRight
Only Right channel have sound.

enumerator kSAI_MonoLeft
Only left channel have sound.

enum _sai_data_order
SAI data order, MSB or LSB.
Values:

enumerator kSAI_DataLSB
LSB bit transferred first

enumerator kSAI_DataMSB
MSB bit transferred first

enum _sai_clock_polarity
SAI clock polarity, active high or low.
Values:

enumerator kSAI_PolarityActiveHigh
Drive outputs on rising edge

enumerator kSAI_PolarityActiveLow
Drive outputs on falling edge

enumerator kSAI_SampleOnFallingEdge
Sample inputs on falling edge

enumerator kSAI_SampleOnRisingEdge
Sample inputs on rising edge

enum _sai_sync_mode
Synchronous or asynchronous mode.
Values:

enumerator kSAI_ModeAsync
Asynchronous mode

enumerator kSAI_ModeSync
Synchronous mode (with receiver or transmit)

enumerator kSAI_ModeSyncWithOtherTx
Synchronous with another SAI transmit

enumerator kSAI_ModeSyncWithOtherRx
Synchronous with another SAI receiver

enum _sai_bclk_source
Bit clock source.
Values:

enumerator kSAI_BclkSourceBusclk
Bit clock using bus clock

enumerator kSAI_BclkSourceMclkOption1
Bit clock MCLK option 1

enumerator kSAI_BclkSourceMclkOption2
Bit clock MCLK option2

enumerator kSAI_BclkSourceMclkOption3
Bit clock MCLK option3

enumerator kSAI_BclkSourceMclkDiv
Bit clock using master clock divider

enumerator kSAI_BclkSourceOtherSai0
Bit clock from other SAI device

enumerator kSAI_BclkSourceOtherSai1
Bit clock from other SAI device

_sai_interrupt_enable_t, The SAI interrupt enable flag
Values:

enumerator kSAI_WordStartInterruptEnable
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorInterruptEnable
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOWarningInterruptEnable
FIFO warning flag, means the FIFO is empty

enumerator kSAI_FIFOErrorInterruptEnable
FIFO error flag

enumerator kSAI_FIFORequestInterruptEnable
FIFO request, means reached watermark

_sai_dma_enable_t, The DMA request sources
Values:

enumerator kSAI_FIFOWarningDMAEnable
FIFO warning caused by the DMA request

enumerator kSAI_FIFORequestDMAEnable
FIFO request caused by the DMA request

_sai_flags, The SAI status flag
Values:

enumerator kSAI_WordStartFlag
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorFlag
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOErrorFlag
FIFO error flag

enumerator kSAI_FIFORequestFlag
FIFO request flag.

enumerator kSAI_FIFOWarningFlag
FIFO warning flag

enum _sai_reset_type
The reset type.
Values:

enumerator kSAI_ResetTypeSoftware
Software reset, reset the logic state

enumerator kSAI_ResetTypeFIFO
FIFO reset, reset the FIFO read and write pointer

enumerator kSAI_ResetAll
All reset.

enum _sai_fifo_packing
The SAI packing mode The mode includes 8 bit and 16 bit packing.
Values:

enumerator kSAI_FifoPackingDisabled
Packing disabled

enumerator kSAI_FifoPacking8bit
8 bit packing enabled

enumerator kSAI_FifoPacking16bit
16bit packing enabled

enum _sai_sample_rate
Audio sample rate.
Values:

enumerator kSAI_SampleRate8KHz
Sample rate 8000 Hz

enumerator kSAI_SampleRate11025Hz
Sample rate 11025 Hz

enumerator kSAI_SampleRate12KHz
Sample rate 12000 Hz

enumerator kSAI_SampleRate16KHz
Sample rate 16000 Hz

enumerator kSAI_SampleRate22050Hz
Sample rate 22050 Hz

enumerator kSAI_SampleRate24KHz
Sample rate 24000 Hz

enumerator kSAI_SampleRate32KHz
Sample rate 32000 Hz

enumerator kSAI_SampleRate44100Hz
Sample rate 44100 Hz

enumerator kSAI_SampleRate48KHz
Sample rate 48000 Hz

enumerator kSAI_SampleRate96KHz
Sample rate 96000 Hz

enumerator kSAI_SampleRate192KHz
Sample rate 192000 Hz

enumerator kSAI_SampleRate384KHz
Sample rate 384000 Hz

enum _sai_word_width
Audio word width.
Values:

enumerator kSAI_WordWidth8bits
Audio data width 8 bits

enumerator kSAI_WordWidth16bits
Audio data width 16 bits

enumerator kSAI_WordWidth24bits
Audio data width 24 bits

enumerator kSAI_WordWidth32bits
Audio data width 32 bits

enum _sai_data_pin_state
sai data pin state definition
Values:

enumerator kSAI_DataPinStateTriState
transmit data pins are tri-stated when slots are masked or channels are disabled

enumerator kSAI_DataPinStateOutputZero
transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

enum _sai_fifo_combine

sai fifo combine mode definition

Values:

enumerator kSAI_FifoCombineDisabled

sai TX/RX fifo combine mode disabled

enumerator kSAI_FifoCombineModeEnabledOnRead

sai TX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnWrite

sai TX fifo combine mode enabled on FIFO write

enumerator kSAI_RxFifoCombineModeEnabledOnWrite

sai RX fifo combine mode enabled on FIFO write

enumerator kSAI_RXFifoCombineModeEnabledOnRead

sai RX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnReadWrite

sai TX/RX fifo combined mode enabled on FIFO read/writes

enum _sai_transceiver_type

sai transceiver type

Values:

enumerator kSAI_Transmitter

sai transmitter

enumerator kSAI_Receiver

sai receiver

enum _sai_frame_sync_len

sai frame sync len

Values:

enumerator kSAI_FrameSyncLenOneBitClk

1 bit clock frame sync len for DSP mode

enumerator kSAI_FrameSyncLenPerWordWidth

Frame sync length decided by word width

typedef enum *_sai_protocol* sai_protocol_t

Define the SAI bus type.

typedef enum *_sai_master_slave* sai_master_slave_t

Master or slave mode.

typedef enum *_sai_mono_stereo* sai_mono_stereo_t
Mono or stereo audio format.

typedef enum *_sai_data_order* sai_data_order_t
SAI data order, MSB or LSB.

typedef enum *_sai_clock_polarity* sai_clock_polarity_t
SAI clock polarity, active high or low.

typedef enum *_sai_sync_mode* sai_sync_mode_t
Synchronous or asynchronous mode.

typedef enum *_sai_bclk_source* sai_bclk_source_t
Bit clock source.

typedef enum *_sai_reset_type* sai_reset_type_t
The reset type.

typedef enum *_sai_fifo_packing* sai_fifo_packing_t
The SAI packing mode The mode includes 8 bit and 16 bit packing.

typedef struct *_sai_config* sai_config_t
SAI user configuration structure.

typedef enum *_sai_sample_rate* sai_sample_rate_t
Audio sample rate.

typedef enum *_sai_word_width* sai_word_width_t
Audio word width.

typedef enum *_sai_data_pin_state* sai_data_pin_state_t
sai data pin state definition

typedef enum *_sai_fifo_combine* sai_fifo_combine_t
sai fifo combine mode definition

typedef enum *_sai_transceiver_type* sai_transceiver_type_t
sai transceiver type

typedef enum *_sai_frame_sync_len* sai_frame_sync_len_t
sai frame sync len

typedef struct *_sai_transfer_format* sai_transfer_format_t
sai transfer format

typedef struct *_sai_master_clock* sai_master_clock_t
master clock configurations

typedef struct *_sai_fifo* sai_fifo_t

sai fifo configurations

typedef struct *_sai_bit_clock* sai_bit_clock_t

sai bit clock configurations

typedef struct *_sai_frame_sync* sai_frame_sync_t

sai frame sync configurations

typedef struct *_sai_serial_data* sai_serial_data_t

sai serial data configurations

typedef struct *_sai_transceiver* sai_transceiver_t

sai transceiver configurations

typedef struct *_sai_transfer* sai_transfer_t

SAI transfer structure.

typedef struct *_sai_handle* sai_handle_t

typedef void (*sai_transfer_callback_t)(I2S_Type *base, *sai_handle_t* *handle, *status_t* status, void *userData)

SAI transfer callback prototype.

MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK

Used to control whether *SAI_RxSetFifoConfig()*/SAI_TxSetFifoConfig() allows a NULL FIFO watermark.

If this macro is set to 0 then *SAI_RxSetFifoConfig()*/SAI_TxSetFifoConfig() will set the watermark to half of the FIFO's depth if passed a NULL watermark.

MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG

Disable implicit channel data configuration within *SAI_TxSetConfig()*/SAI_RxSetConfig().

Use this macro to control whether *SAI_RxSetConfig()*/SAI_TxSetConfig() will attempt to implicitly configure the channel data. By channel data we mean the startChannel, channelMask, endChannel, and channelNums fields from the sai_transciever_t structure. By default, *SAI_TxSetConfig()*/SAI_RxSetConfig() will attempt to compute these fields, which may not be desired in cases where the user wants to set them before the call to said functions.

SAI_XFER_QUEUE_SIZE

SAI transfer queue size, user can refine it according to use case.

FSL_SAI_HAS_FIFO_EXTEND_FEATURE

sai fifo feature

struct _sai_config

#include <fsl_sai.h> SAI user configuration structure.

Public Members

sai_protocol_t protocol
Audio bus protocol in SAI

sai_sync_mode_t syncMode
SAI sync mode, control Tx/Rx clock sync

bool mclkOutputEnable
Master clock output enable, true means master clock divider enabled

sai_bclk_source_t bclkSource
Bit Clock source

sai_master_slave_t masterSlave
Master or slave

struct __sai__transfer__format
#include <fsl_sai.h> sai transfer format

Public Members

uint32_t sampleRate__Hz
Sample rate of audio data

uint32_t bitWidth
Data length of audio data, usually 8/16/24/32 bits

sai_mono_stereo_t stereo
Mono or stereo

uint32_t masterClockHz
Master clock frequency in Hz

uint8_t watermark
Watermark value

uint8_t channel
Transfer start channel

uint8_t channelMask
enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

sai_protocol_t protocol
: Which audio protocol used

bool isFrameSyncCompact
: True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

struct _sai_master_clock
: *#include <fsl_sai.h>* master clock configurations

Public Members

bool mclkOutputEnable
: master clock output enable

uint32_t mclkHz
: target mclk frequency

uint32_t mclkSourceClkHz
: mclk source frequency

struct _sai_fifo
: *#include <fsl_sai.h>* sai fifo configurations

Public Members

bool fifoContinueOneError
: fifo continues when error occur

sai_fifo_combine_t fifoCombine
: fifo combine mode

sai_fifo_packing_t fifoPacking
: fifo packing mode

uint8_t fifoWatermark
: fifo watermark

struct _sai_bit_clock
: *#include <fsl_sai.h>* sai bit clock configurations

Public Members

bool bclkSrcSwap
: bit clock source swap

bool bclkInputDelay

bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .

sai_clock_polarity_t bclkPolarity

bit clock polarity

sai_bclk_source_t bclkSource

bit Clock source

struct __sai__frame__sync

#include <fsl_sai.h> sai frame sync configurations

Public Members

uint8_t frameSyncWidth

frame sync width in number of bit clocks

bool frameSyncEarly

TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

bool frameSyncGenerateOnDemand

internal frame sync is generated when FIFO waring flag is clear

sai_clock_polarity_t frameSyncPolarity

frame sync polarity

struct __sai__serial__data

#include <fsl_sai.h> sai serial data configurations

Public Members

sai_data_pin_state_t dataMode

sai data pin state when slots masked or channel disabled

sai_data_order_t dataOrder

configure whether the LSB or MSB is transmitted first

uint8_t dataWord0Length

configure the number of bits in the first word in each frame

uint8_t dataWordNLength

configure the number of bits in the each word in each frame, except the first word

uint8_t dataWordLength
used to record the data length for dma transfer

uint8_t dataFirstBitShifted
Configure the bit index for the first bit transmitted for each word in the frame

uint8_t dataWordNum
configure the number of words in each frame

uint32_t dataMaskedWord
configure whether the transmit word is masked

struct __sai__transceiver
#include <fsl_sai.h> sai transceiver configurations

Public Members

sai_serial_data_t serialData
serial data configurations

sai_frame_sync_t frameSync
ws configurations

sai_bit_clock_t bitClock
bit clock configurations

sai_fifo_t fifo
fifo configurations

sai_master_slave_t masterSlave
transceiver is master or slave

sai_sync_mode_t syncMode
transceiver sync mode

uint8_t startChannel
Transfer start channel

uint8_t channelMask
enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

struct __sai__transfer
#include <fsl_sai.h> SAI transfer structure.

Public Members

uint8_t *data
Data start address to transfer.

size_t dataSize
Transfer size.

struct _sai_handle
#include <fsl_sai.h> SAI handle structure.

Public Members

I2S_Type *base
base address

uint32_t state
Transfer status

sai_transfer_callback_t callback
Callback function called at transfer event

void *userData
Callback parameter passed to callback function

uint8_t bitWidth
Bit width for transfer, 8/16/24/32 bits

uint8_t channel
Transfer start channel

uint8_t channelMask
enabled channel mask value, refernece _sai_channel_mask

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

sai_transfer_t saiQueue[(4U)]
Transfer queue storing queued transfer

size_t transferSize[(4U)]
Data bytes need to transfer

volatile uint8_t queueUser
Index for user to queue transfer

volatile uint8_t queueDriver

Index for driver to get the transfer data and size

uint8_t watermark

Watermark value

2.80 SAI EDMA Driver

void SAI_TransferTxCreateHandleEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_edma_callback_t* callback, void *userData, *edma_handle_t* *txDmaHandle)

Initializes the SAI eDMA handle.

This function initializes the SAI master DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- txDmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void SAI_TransferRxCreateHandleEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_edma_callback_t* callback, void *userData, *edma_handle_t* *rxDmaHandle)

Initializes the SAI Rx eDMA handle.

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- rxDmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void SAI_TransferSetInterleaveType(*sai_edma_handle_t* *handle, *sai_edma_interleave_t* interleaveType)

Initializes the SAI interleave type.

This function initializes the SAI DMA handle member interleaveType, it shall be called only when application would like to use type kSAI_EDMAInterleavePerChannelBlock, since the default interleaveType is kSAI_EDMAInterleavePerChannelSample always

Parameters

- handle – SAI eDMA handle pointer.

- interleaveType – Multi channel interleave type.

void SAI_TransferTxSetConfigEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transceiver_t* *saiConfig)

Configures the SAI Tx.

Note: SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of sai_transceiver_t with the corresponding values of _sai_channel_mask enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnWrite to the fifoCombine member of sai_fifo_combine_t which is a member of sai_transceiver_t. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_
↪Channel1Mask);
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnWrite;
SAI_TransferTxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

void SAI_TransferRxSetConfigEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transceiver_t* *saiConfig)

Configures the SAI Rx.

Note: SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of sai_transceiver_t with the corresponding values of _sai_channel_mask enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnRead to the fifoCombine member of sai_fifo_combine_t which is a member of sai_transceiver_t. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_
↪Channel1Mask);
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnRead;
SAI_TransferRxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

status_t SAI_TransferSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI transfer using DMA.

This function support multi channel transfer,

a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers

b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Note: This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

Return values

- kStatus_Success – Start a SAI eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_TxBusy – SAI is busy sending data.

status_t SAI_TransferReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI receive using eDMA.

This function support multi channel transfer,

a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers

b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Note: This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemainingBytes to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a SAI eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_RxBusy – SAI is busy receiving data.

status_t SAI_TransferSendLoopEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer, uint32_t loopTransferCount)

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function SAI_TransferAbortSendEDMA to stop the loop transfer.

Note: This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).
- loopTransferCount – the counts of xfer array.

Return values

- kStatus_Success – Start a SAI eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.

status_t SAI_TransferReceiveLoopEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer, uint32_t loopTransferCount)

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function SAI_TransferAbortReceiveEDMA to stop the loop transfer.

Note: This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).
- loopTransferCount – the counts of xfer array.

Return values

- kStatus_Success – Start a SAI eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.

void SAI_TransferTerminateSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSendEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceiveEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferAbortSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Aborts a SAI transfer using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateSendEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferAbortReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Aborts a SAI receive using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateReceiveEDMA.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.

status_t SAI_TransferGetSendCountEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, size_t *count)

Gets byte count sent by SAI.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- count – Bytes count sent by SAI.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

status_t SAI_TransferGetReceiveCountEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, size_t *count)

Gets byte count received by SAI.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.
- count – Bytes count received by SAI.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

uint32_t SAI_TransferGetValidTransferSlotsEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Gets valid transfer slot.

This function can be used to query the valid transfer request slot that the application can submit. It should be called in the critical section, that means the application could call it in the corresponding callback function or disable IRQ before calling it in the application, otherwise, the returned value may not correct.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.

Return values

valid – slot count that application submit.

FSL_SAI_EDMA_DRIVER_VERSION

Version 2.7.4

enum _sai_edma_interleave

sai interleave type

Values:

enumerator kSAI_EDMAInterleavePerChannelSample

enumerator kSAI_EDMAInterleavePerChannelBlock

typedef struct *sai_edma_handle* sai_edma_handle_t

typedef void (*sai_edma_callback_t)(I2S_Type *base, *sai_edma_handle_t* *handle, *status_t* status, void *userData)

SAI eDMA transfer callback function for finish and error.

typedef enum *_sai_edma_interleave* sai_edma_interleave_t

sai interleave type

MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL

the SAI enable position When calling SAI_TransferReceiveEDMA

MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL
the SAI enable position When calling SAI_TransferSendEDMA

struct sai_edma_handle
#include <fsl_sai_edma.h> SAI DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle
DMA handler for SAI send

uint8_t nbytes
eDMA minor byte transfer count initially configured.

uint8_t bytesPerFrame
Bytes in a frame

uint8_t channelMask
Enabled channel mask value, reference _sai_channel_mask

uint8_t channelNums
total enabled channel nums

uint8_t channel
Which data channel

uint8_t count
The transfer data count in a DMA request

uint32_t state
Internal state for SAI eDMA transfer

sai_edma_callback_t callback
Callback for users while transfer finish or error occurs

void *userData
User callback parameter

uint8_t tcd[((4U) + 1U) * sizeof(*edma_tcd_t*)]
TCD pool for eDMA transfer.

sai_transfer_t saiQueue[(4U)]
Transfer queue storing queued transfer.

size_t transferSize[(4U)]
Data bytes need to transfer

sai_edma_interleave_t interleaveType
Transfer interleave type

volatile uint8_t queueUser
Index for user to queue transfer.

volatile uint8_t queueDriver
Index for driver to get the transfer data and size

2.81 Sbloader

enum _sectionType
sb3 section definitions
section type
Values:

enumerator kSectionNone
end or invalid

enumerator kSectionDataRange

enumerator kSectionDiffUpdate

enumerator kSectionDDRConfig

enumerator kSectionRegister

Values:

enumerator kFwVerChk_Id_none

enumerator kFwVerChk_Id_nonsecure

enumerator kFwVerChk_Id_secure

enum _loader_command_sb3
loader command enum
Values:

enumerator kSB3_CmdInvalid

enumerator kSB3_CmdErase

enumerator kSB3_CmdLoad

enumerator kSB3_CmdExecute

enumerator kSB3_CmdCall

enumerator kSB3_CmdProgramFuse

enumerator kSB3_CmdProgramIFR

enumerator kSB3_CmdLoadCmac

enumerator kSB3_CmdCopy

enumerator kSB3_CmdLoadHashLocking

enumerator kSB3_CmdLoadKeyBlob

enumerator kSB3_CmdConfigMem

enumerator kSB3_CmdFillMem

enumerator kSB3_CmdFwVerCheck

enum kb_operation_t

Details of the operation to be performed by the ROM.

The *kRomAuthenticateImage* operation requires the entire signed image to be available to the application.

Values:

enumerator kRomAuthenticateImage

Authenticate a signed image.

enumerator kRomLoadImage

Load SB file.

enumerator kRomOperationCount

typedef uint8_t chunk_v3_t[16]

typedef struct *_ldr_buf* ldr_buf_t

typedef struct *_ldr_Context_v3* ldr_Context_v3_t

Provides forward reference to the loader context definition.

typedef *status_t* (*pLdrFnc_v3_t)(*ldr_Context_v3_t* *content)

Function pointer definition for all loader action functions.

typedef enum *_sectionType* section_type_t

sb3 section definitions

section type

typedef struct *range_header* sb3_data_range_header_t

section data range structure

typedef struct *range_header_expansion* sb3_data_range_expansion_t

typedef struct *copy_memory_expansion* sb3_copy_memory_expansion_t

typedef struct *copy* sb3_copy_memory_t

typedef struct *load_keyblob* sb3_load_keyblob_t

typedef struct *fill_memory_expansion* sb3_fill_memory_expansion_t

typedef struct *fill_memory* sb3_fill_memory_t

typedef struct *config_memory* sb3_config_memory_t

typedef struct *fw_ver_check* sb3_fw_ver_check_t

typedef struct *section_header* sb3_section_header_t

sb3 DATA section header format

typedef enum *_loader_command_sb3* sb3_cmd_t

loader command enum

SB3_BYTES_PER_CHUNK

Defines the number of bytes in a cipher block (chunk). This is dictated by the encryption algorithm.

SB3_DATA_RANGE_HEADER_FLAGS_ERASE_MASK

bit 0

SB3_DATA_RANGE_HEADER_FLAGS_LOAD_MASK

bit 1

SBLOADER_V3_CMD_SET_ALL

The all of the allowed command.

SBLOADER_V3_CMD_SET_IN_ISP_MODE

The allowed command set in ISP mode.

SBLOADER_V3_CMD_SET_IN_REC_MODE
The allowed command set in recovery mode.

SB3_DATA_BUFFER_SIZE_IN_BYTE

struct _ldr_buf
#include <fsl_sbloader_v3.h>

struct range_header
#include <fsl_sbloader_v3.h> section data range structure

struct range_header_expansion
#include <fsl_sbloader_v3.h>

struct copy_memory_expansion
#include <fsl_sbloader_v3.h>

struct copy
#include <fsl_sbloader_v3.h>

struct load_keyblob
#include <fsl_sbloader_v3.h>

struct fill_memory_expansion
#include <fsl_sbloader_v3.h>

Public Members

uint32_t pattern
word to be used as pattern

struct fill_memory
#include <fsl_sbloader_v3.h>

struct config_memory
#include <fsl_sbloader_v3.h>

Public Members

uint32_t address
address of config blob

struct fw_ver_check
#include <fsl_sbloader_v3.h>

struct section_header

#include <fsl_sbloader_v3.h> sb3 DATA section header format

struct kb_region_t

#include <fsl_sbloader_v3.h> Memory region definition.

struct kb_load_sb_t

#include <fsl_sbloader_v3.h>

struct kb_authenticate_t

#include <fsl_sbloader_v3.h>

struct kb_options_t

#include <fsl_sbloader_v3.h>

Public Members

uint32_t version

Should be set to #kKbootApiVersion.

uint8_t *buffer

Caller-provided buffer used by Kboot.

struct _ldr_Context_v3

#include <fsl_sbloader_v3.h> Loader context definition.

Public Members

pLdrFnc_v3_t Action

pointer to loader action function

uint32_t block_size

size of each block in bytes

uint32_t block_data_size

data size in bytes (NBOOT_SB3_CHUNK_SIZE_IN_BYTES)

uint32_t block_data_total

data max size in bytes (block_size * data_size

uint32_t block_buffer_size

block0 and block size

uint32_t processedBlocks

will be used for both block0 and blockx

bool in_data_block
: data block offset in a block. in progress of handling a data block within a block

bool in_data_section
: in progress of handling a data section within a data block

bool in_data_range
: in progress of handling a data range within a data section

uint32_t commandSet
: support command set during sb file handling

uint8_t data_buffer[(*MAX*(128, NBOOT_KEY_BLOB_SIZE_IN_BYTE_MAX))]
: temporary data buffer

kb_options_t fromAPI
: options from ROM API

union ___unnamed94___

Public Members

kb_authenticate_t authenticate
: Settings for #kKbootAuthenticate operation.

kb_load_sb_t loadSB
: Settings for #kKbootLoadSB operation.

2.82 SCTimer: SCTimer/PWM (SCT)

status_t SCTIMER_Init(SCT_Type *base, const *sctimer_config_t* *config)
: Ungates the SCTimer clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application using the SCTimer driver.

Parameters

- base – SCTimer peripheral base address
- config – Pointer to the user configuration structure.

Returns
kStatus_Success indicates success; Else indicates failure.

void SCTIMER_Deinit(SCT_Type *base)
: Gates the SCTimer clock.

Parameters

- base – SCTimer peripheral base address

void SCTIMER_GetDefaultConfig(*sctimer_config_t* *config)

Fills in the SCTimer configuration structure with the default settings.

The default values are:

```
config->enableCounterUnify = true;
config->clockMode = kSCTIMER_System_ClockMode;
config->clockSelect = kSCTIMER_Clock_On_Rise_Input_0;
config->enableBidirection_l = false;
config->enableBidirection_h = false;
config->prescale_l = 0U;
config->prescale_h = 0U;
config->outInitState = 0U;
config->inputsync  = 0xFU;
```

Parameters

- config – Pointer to the user configuration structure.

status_t SCTIMER_SetupPwm(SCT_Type *base, const *sctimer_pwm_signal_param_t* *pwmParams, *sctimer_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, uint32_t *event)

Configures the PWM signal parameters.

Deprecated:

Use *SCTIMER_SetupSharedPeriodPwm()* instead. That function configures one channel per call and reuses a single shared period/limit event across same-frequency channels (N + 1 events for N channels instead of 2 * N), and returns the pulse event explicitly so the duty cycle can be updated with *SCTIMER_UpdatePwmDutycycleByEvent()*. This function remains functional for backward compatibility.

Call this function to configure the PWM signal period, mode, duty cycle, and edge. This function will create 2 events; one of the events will trigger on match with the pulse value and the other will trigger when the counter matches the PWM period. The PWM period event is also used as a limit event to reset the counter or change direction. Both events are enabled for the same state. The state number can be retrieved by calling the function SCTIMER_GetCurrentStateNumber(). The counter is set to operate as one 32-bit counter (unify bit is set to 1). The counter operates in bi-directional mode when generating a center-aligned PWM.

Note: When setting PWM output from multiple output pins, they all should use the same PWM mode i.e all PWM's should be either edge-aligned or center-aligned. When using this API, the PWM signal frequency of all the initialized channels must be the same. Otherwise all the initialized channels' PWM signal frequency is equal to the last call to the API's pwmFreq_Hz.

Parameters

- base – SCTimer peripheral base address
- pwmParams – PWM parameters to configure the output
- mode – PWM operation mode, options available in enumeration *sctimer_pwm_mode_t*
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – SCTimer counter clock in Hz

- event – Pointer to a variable where the PWM period event number is stored

Returns

kStatus_Success on success kStatus_Fail If we have hit the limit in terms of number of events created or if an incorrect PWM dutycylce is passed in.

void SCTIMER_UpdatePwmDutycycle(SCT_Type *base, *sctimer_out_t* output, uint8_t dutyCyclePercent, uint32_t event)

Updates the duty cycle of an active PWM signal.

Deprecated:

Use *SCTIMER_UpdatePwmDutycycleByEvent()* instead. That function takes the period and pulse events explicitly (it does not assume the pulse event is event + 1), so it works with channels created by *SCTIMER_SetupSharedPeriodPwm()*, and it can update immediately or glitchlessly at the next period. This function remains functional for backward compatibility.

Before calling this function, the counter is set to operate as one 32-bit counter (unify bit is set to 1).

Parameters

- base – SCTimer peripheral base address
- output – The output to configure
- dutyCyclePercent – New PWM pulse width; the value should be between 1 to 100
- event – Event number associated with this PWM signal. This was returned to the user by the function *SCTIMER_SetupPwm()*.

status_t SCTIMER_SetupSharedPeriodPwm(SCT_Type *base, const *sctimer_pwm_signal_param_t* *pwmParam, *sctimer_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, uint32_t *periodEvent, uint32_t *pulseEvent)

Configures one PWM channel that shares an auto-detected period event across calls.

Unlike *SCTIMER_SetupPwm()* (which spends two events per channel and recreates a period event each call), this function configures a single channel and reuses **one** shared period/limit event for all same-frequency channels. On each call it inspects the unified-counter limit mask (LIMIT[LIMMSK_L]) for an existing period/limit event:

- none present -> it creates the shared period/limit event (1 period + 1 pulse event), or
- one present whose period (its MATCH value) and alignment (CTRL[BIDIR_L] set <=> center-aligned) match the requested pwmFreq_Hz/mode -> it reuses that event and only creates this channel's pulse event (1 pulse event), or
- one present whose period or alignment differs -> it returns kStatus_InvalidArgument without creating or modifying anything (two periods cannot share one unified counter).

So K same-frequency channels built by K calls consume K + 1 events / K + 1 match registers instead of 2 * K. A period/limit event left by a prior *SCTIMER_SetupPwm()* at the same period/alignment is also reusable. The counter must already be configured as one unified 32-bit counter (CONFIG[UNIFY] = 1).

Note: Period detection assumes the shared PWM period/limit is the lowest-numbered unified-counter limit event (the only limit events created by this function and *SCTIMER_SetupPwm()*). If the application created other unified-counter limit events before

calling this function, detection may match the wrong event; in that case create the PWM channels before any unrelated limit events.

Note: Each call configures exactly one channel and is atomic for that channel only (on failure it creates nothing). Building a multi-channel group is a sequence of independent calls; this API does NOT roll the whole group back if a later call fails.

Note: The channel is enabled in the current state only. In a multi-state design the caller must re-enable the returned periodEvent and each channel's pulseEvent in every state that should run the group (via *SCTIMER_ScheduleEvent()*). Because the period event is shared, failing to re-schedule it in a state disrupts ALL channels in that state, not just one.

Parameters

- base – SCTimer peripheral base address
- pwmParam – Pointer to this channel's PWM parameters (output, level, duty)
- mode – PWM operation mode (all sharing channels must use the same mode), see *sctimer_pwm_mode_t*
- pwmFreq_Hz – Common PWM signal frequency in Hz (all sharing channels must use the same frequency)
- srcClock_Hz – SCTimer counter clock in Hz
- periodEvent – Pointer to a variable where the shared period event number (new or reused) is stored
- pulseEvent – Pointer to a variable where this channel's pulse event number is stored

Returns

kStatus_Success on success kStatus_InvalidArgument if arguments are invalid, the counter is not in unified mode, or an existing shared period event has a different frequency/alignment kStatus_OutOfRange if the event or match-register budget would be exceeded

void SCTIMER_UpdatePwmDutycycleByEvent(SCT_Type *base, *sctimer_out_t* output, uint8_t dutyCyclePercent, uint32_t periodEvent, uint32_t pulseEvent, *sctimer_pwm_update_mode_t* updateMode)

Updates the duty cycle of a PWM signal using explicit period and pulse events.

Unlike *SCTIMER_UpdatePwmDutycycle()*, this function does not assume the pulse event is periodEvent + 1, so it works for channels created by *SCTIMER_SetupSharedPeriodPwm()* where the pulse events are not adjacent to the shared period event. The counter must be in unified 32-bit mode.

Parameters

- base – SCTimer peripheral base address
- output – The output to configure
- dutyCyclePercent – New PWM pulse width; the value should be between 0 and 100
- periodEvent – The shared period event (as returned by *SCTIMER_SetupSharedPeriodPwm()*)

- pulseEvent – The channel's pulse event (as returned by *SCTIMER_SetupSharedPeriodPwm()*)
- updateMode – When the new duty cycle takes effect, see *sctimer_pwm_update_mode_t*

status_t SCTIMER_SetupComplementaryPwm(SCT_Type *base, *sctimer_out_t* outHigh, *sctimer_out_t* outLow, uint8_t dutyCyclePercent, uint32_t deadTimeTicks, *sctimer_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, *sctimer_complementary_pwm_handle_t* *handle)

Configures a complementary PWM output pair with programmable dead time.

Builds a high-side/low-side complementary pair on the unified 32-bit counter from one duty value plus a dead time expressed in SCT counter clock ticks. Dead time is inserted by delaying each output's turn-on by deadTimeTicks while keeping turn-off immediate, which guarantees a non-overlap (dead-time) gap at both switching edges so the two outputs are never simultaneously active (no shoot-through).

The shared period event also limits (resets) the counter. The returned handle is used to update the pair's duty cycle later with *SCTIMER_UpdateComplementaryPwmDutycycle()*.

Note: Dead time is inserted by delaying the relevant turn-on edge - the high side in edge-aligned mode, the low side in center-aligned mode - so that output's active width shrinks by deadTimeTicks (the usual cost of dead time).

Note: For a switching duty the valid range is 0 < duty < 100 (needs deadTimeTicks < duty and duty + deadTimeTicks < period); deadTimeTicks = 0 is allowed (strict complementary). 0 % and 100 % are accepted as constant complementary levels (high-side fully off / fully on): the pair is driven to static opposite levels with no switching and no dead-time window. A both-off fault state is not expressible as a duty - use a separate output force/disable path for that.

Note: Call this function once per complementary pair. To drive several pairs on the same SCTimer (for example the three half-bridges of a 3-phase inverter), call it once for each pair — but all pairs MUST use the same pwmFreq_Hz and the same mode, because they share one unified counter (a single count direction and period). Mixing frequencies or alignments across pairs creates conflicting limit events on the counter and produces an incorrect waveform.

Parameters

- base – SCTimer peripheral base address
- outHigh – High-side output pin
- outLow – Low-side output pin (complement of outHigh)
- dutyCyclePercent – High-side duty cycle, value should be between 0 and 100
- deadTimeTicks – Dead time in SCT counter clock ticks, inserted at both edges; 0 = no dead time (strict complementary)
- mode – PWM operation mode, see *sctimer_pwm_mode_t*
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – SCTimer counter clock in Hz

- handle – Pointer to a handle that receives the pair's event numbers and parameters

Returns

kStatus_Success on success kStatus_InvalidArgument if arguments are invalid, the counter is not unified, or the dead time does not fit the requested duty/period without causing overlap kStatus_OutOfRange if the event or match-register budget would be exceeded

status_t SCTIMER_UpdateComplementaryPwmDutycycle(SCT_Type *base, const *sctimer_complementary_pwm_handle_t* *handle, uint8_t dutyCyclePercent, *sctimer_pwm_update_mode_t* updateMode)

Updates the duty cycle of a complementary PWM pair, preserving dead time.

Recomputes both moving switching edges (high-side turn-off and low-side turn-on) from the new duty cycle while keeping the dead-time offset, and writes them so the no-overlap guarantee holds across the update. In kSCTIMER_UpdateOnNextPeriod mode both edges reload together at the next counter cycle, so the dead-time relationship is updated atomically and glitchlessly.

Note: A switching duty must keep the dead time inside the period (deadTimeTicks < duty and duty + deadTimeTicks < period). 0 % and 100 % are accepted as constant complementary levels: the pair is driven to static opposite levels (no switching, no dead time). Switching between a constant level and a switching duty briefly halts the counter to re-bind the outputs. A both-off fault state is not a duty - use a separate output force/disable path.

Note: updateMode is honored only for a switching duty (0 < duty < 100). A 0 % or 100 % update is always applied immediately (counter briefly halted), regardless of updateMode, because a constant level cannot be loaded through the glitchless reload-register (MATCHREL) path: it requires detaching the events and forcing the output levels, which needs a halt. kSCTIMER_UpdateOnNextPeriod therefore does not defer a 0 %/100 % change to the next period boundary.

Parameters

- base – SCTimer peripheral base address
- handle – Handle returned by *SCTIMER_SetupComplementaryPwm()*
- dutyCyclePercent – New high-side duty cycle, value should be between 0 and 100
- updateMode – When the new duty cycle takes effect, see *sctimer_pwm_update_mode_t*. Ignored for a 0 %/100 % update, which is always immediate; see the note above.

Returns

kStatus_Success on success kStatus_InvalidArgument if the dead time does not fit the requested duty/period

static inline void SCTIMER_EnableInterrupts(SCT_Type *base, uint32_t mask)

Enables the selected SCTimer interrupts.

Parameters

- base – SCTimer peripheral base address

- mask – The interrupts to enable. This is a logical OR of members of the enumeration *sctimer_interrupt_enable_t*

static inline void SCTIMER_DisableInterrupts(SCT_Type *base, uint32_t mask)

Disables the selected SCTimer interrupts.

Parameters

- base – SCTimer peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *sctimer_interrupt_enable_t*

static inline uint32_t SCTIMER_GetEnabledInterrupts(SCT_Type *base)

Gets the enabled SCTimer interrupts.

Parameters

- base – SCTimer peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *sctimer_interrupt_enable_t*

static inline uint32_t SCTIMER_GetStatusFlags(SCT_Type *base)

Gets the SCTimer status flags.

Parameters

- base – SCTimer peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration *sctimer_status_flags_t*

static inline void SCTIMER_ClearStatusFlags(SCT_Type *base, uint32_t mask)

Clears the SCTimer status flags.

Parameters

- base – SCTimer peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration *sctimer_status_flags_t*

static inline void SCTIMER_StartTimer(SCT_Type *base, uint32_t countertoStart)

Starts the SCTimer counter.

Note: In 16-bit mode, we can enable both Counter_L and Counter_H, In 32-bit mode, we only can select Counter_U.

Parameters

- base – SCTimer peripheral base address
- countertoStart – The SCTimer counters to enable. This is a logical OR of members of the enumeration *sctimer_counter_t*.

static inline void SCTIMER_StopTimer(SCT_Type *base, uint32_t countertoStop)

Halts the SCTimer counter.

Parameters

- base – SCTimer peripheral base address

- countertoStop – The SCTimer counters to stop. This is a logical OR of members of the enumeration *sctimer_counter_t*.

status_t SCTIMER_CreateAndScheduleEvent(SCT_Type *base, *sctimer_event_t* howToMonitor, uint32_t matchValue, uint32_t whichIO, *sctimer_counter_t* whichCounter, uint32_t *event)

Create an event that is triggered on a match or IO and schedule in current state.

This function will configure an event using the options provided by the user. If the event type uses the counter match, then the function will set the user provided match value into a match register and put this match register number into the event control register. The event is enabled for the current state and the event number is increased by one at the end. The function returns the event number; this event number can be used to configure actions to be done when this event is triggered.

Parameters

- base – SCTimer peripheral base address
- howToMonitor – Event type; options are available in the enumeration *sctimer_interrupt_enable_t*
- matchValue – The match value that will be programmed to a match register
- whichIO – The input or output that will be involved in event triggering. This field is ignored if the event type is "match only"
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Pointer to a variable where the new event number is stored

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of events created or if we have reached the limit in terms of number of match registers

void SCTIMER_ScheduleEvent(SCT_Type *base, uint32_t event)

Enable an event in the current state.

This function will allow the event passed in to trigger in the current state. The event must be created earlier by either calling the function *SCTIMER_SetupPwm()* or function *SCTIMER_CreateAndScheduleEvent()* .

Parameters

- base – SCTimer peripheral base address
- event – Event number to enable in the current state

status_t SCTIMER_IncreaseState(SCT_Type *base)

Increase the state by 1.

All future events created by calling the function *SCTIMER_ScheduleEvent()* will be enabled in this new state.

Parameters

- base – SCTimer peripheral base address

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of states used

uint32_t SCTIMER_GetCurrentState(SCT_Type *base)

Provides the current state.

User can use this to set the next state by calling the function *SCTIMER_SetupNextStateAction()*.

Parameters

- base – SCTimer peripheral base address

Returns

The current state

static inline void SCTIMER_SetCounterState(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t state)

Set the counter current state.

The function is to set the state variable bit field of STATE register. Writing to the STATE_L, STATE_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- state – The counter current state number (only support range from 0~31).

static inline uint16_t SCTIMER_GetCounterState(SCT_Type *base, *sctimer_counter_t* whichCounter)

Get the counter current state value.

The function is to get the state variable bit field of STATE register.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns

The the counter current state value.

status_t SCTIMER_SetupCaptureAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t *captureRegister, uint32_t event)

Setup capture of the counter value on trigger of a selected event.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- captureRegister – Pointer to a variable where the capture register number will be returned. User can read the captured value from this register when the specified event is triggered.
- event – Event number that will trigger the capture

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of match/capture registers available

void SCTIMER_SetCallback(SCT_Type *base, *sctimer_event_callback_t* callback, uint32_t event)

Receive noticification when the event trigger an interrupt.

If the interrupt for the event is enabled by the user, then a callback can be registered which will be invoked when the event is triggered

Parameters

- base – SCTimer peripheral base address
- event – Event number that will trigger the interrupt
- callback – Function to invoke when the event is triggered

static inline void SCTIMER_SetupStateLdMethodAction(SCT_Type *base, uint32_t event, bool fgLoad)

Change the load method of transition to the specified state.

Change the load method of transition, it will be triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- event – Event number that will change the method to trigger the state transition
- fgLoad – The method to load highest-numbered event occurring for that state to the STATE register.
  - true: Load the STATEV value to STATE when the event occurs to be the next state.
  - false: Add the STATEV value to STATE when the event occurs to be the next state.

static inline void SCTIMER_SetupNextStateActionwithLdMethod(SCT_Type *base, uint32_t nextState, uint32_t event, bool fgLoad)

Transition to the specified state with Load method.

This transition will be triggered by the event number that is passed in by the user, the method decide how to load the highest-numbered event occurring for that state to the STATE register.

Parameters

- base – SCTimer peripheral base address
- nextState – The next state SCTimer will transition to
- event – Event number that will trigger the state transition
- fgLoad – The method to load the highest-numbered event occurring for that state to the STATE register.
  - true: Load the STATEV value to STATE when the event occurs to be the next state.
  - false: Add the STATEV value to STATE when the event occurs to be the next state.

static inline void SCTIMER_SetupNextStateAction(SCT_Type *base, uint32_t nextState, uint32_t event)

Transition to the specified state.

Deprecated:

Do not use this function. It has been superceded by *SCTIMER_SetupNextStateActionwithLdMethod*

This transition will be triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- nextState – The next state SCTimer will transition to
- event – Event number that will trigger the state transition

static inline void SCTIMER_SetupEventActiveDirection(SCT_Type *base, *sctimer_event_active_direction_t* activeDirection, uint32_t event)

Setup event active direction when the counters are operating in BIDIR mode.

Parameters

- base – SCTimer peripheral base address
- activeDirection – Event generation active direction, see *sctimer_event_active_direction_t*.
- event – Event number that need setup the active direction.

static inline void SCTIMER_SetupOutputSetAction(SCT_Type *base, uint32_t whichIO, uint32_t event)

Set the Output.

This output will be set when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichIO – The output to set
- event – Event number that will trigger the output change

static inline void SCTIMER_SetupOutputClearAction(SCT_Type *base, uint32_t whichIO, uint32_t event)

Clear the Output.

This output will be cleared when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichIO – The output to clear
- event – Event number that will trigger the output change

void SCTIMER_SetupOutputToggleAction(SCT_Type *base, uint32_t whichIO, uint32_t event)

Toggle the output level.

This change in the output level is triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- whichIO – The output to toggle
- event – Event number that will trigger the output change

static inline void SCTIMER_SetupCounterLimitAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t event)

Limit the running counter.

The counter is limited when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to be limited

static inline void SCTIMER_SetupCounterStopAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t event)

Stop the running counter.

The counter is stopped when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to be stopped

static inline void SCTIMER_SetupCounterStartAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t event)

Re-start the stopped counter.

The counter will re-start when the event number that is passed in by the user is triggered.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to re-start

static inline void SCTIMER_SetupCounterHaltAction(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t event)

Halt the running counter.

The counter is disabled (halted) when the event number that is passed in by the user is triggered. When the counter is halted, all further events are disabled. The HALT condition can only be removed by calling the *SCTIMER_StartTimer()* function.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- event – Event number that will trigger the counter to be halted

static inline void SCTIMER_SetupDmaTriggerAction(SCT_Type *base, uint32_t dmaNumber, uint32_t event)

Generate a DMA request.

DMA request will be triggered by the event number that is passed in by the user.

Parameters

- base – SCTimer peripheral base address
- dmaNumber – The DMA request to generate
- event – Event number that will trigger the DMA request

static inline void SCTIMER_SetCOUNTValue(SCT_Type *base, *sctimer_counter_t* whichCounter, uint32_t value)

Set the value of counter.

The function is to set the value of Count register, Writing to the COUNT_L, COUNT_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- value – the counter value update to the COUNT register.

static inline uint32_t SCTIMER_GetCOUNTValue(SCT_Type *base, *sctimer_counter_t* whichCounter)

Get the value of counter.

The function is to read the value of Count register, software can read the counter registers at any time..

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns

The value of counter selected.

static inline void SCTIMER_SetEventInState(SCT_Type *base, uint32_t event, uint32_t state)

Set the state mask bit field of EV_STATE register.

Parameters

- base – SCTimer peripheral base address
- event – The EV_STATE register be set.
- state – The state value in which the event is enabled to occur.

static inline void SCTIMER_ClearEventInState(SCT_Type *base, uint32_t event, uint32_t state)

Clear the state mask bit field of EV_STATE register.

Parameters

- base – SCTimer peripheral base address
- event – The EV_STATE register be clear.
- state – The state value in which the event is disabled to occur.

static inline bool SCTIMER_GetEventInState(SCT_Type *base, uint32_t event, uint32_t state)

Get the state mask bit field of EV_STATE register.

Note: This function is to check whether the event is enabled in a specific state.

Parameters

- base – SCTimer peripheral base address
- event – The EV_STATE register be read.
- state – The state value.

Returns
The the state mask bit field of EV_STATE register.

- true: The event is enable in state.
- false: The event is disable in state.

static inline uint32_t SCTIMER_GetCaptureValue(SCT_Type *base, *sctimer_counter_t* whichCounter, uint8_t capChannel)

Get the value of capture register.

This function returns the captured value upon occurrence of the events selected by the corresponding Capture Control registers occurred.

Parameters

- base – SCTimer peripheral base address
- whichCounter – SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
- capChannel – SCTimer capture register of capture channel.

Returns
The SCTimer counter value at which this register was last captured.

void SCTIMER_EventHandleIRQ(SCT_Type *base)

SCTimer interrupt handler.

Parameters

- base – SCTimer peripheral base address.

void SCTIMER_DriverIRQHandler(uint32_t instance)

Common IRQ handler for SCTimer, which can be used for SCTimer IRQ remapping.

Parameters

- instance – SCTimer instance number.

FSL_SCTIMER_DRIVER_VERSION

Version

enum _sctimer_pwm_mode

SCTimer PWM operation modes.

Values:

enumerator kSCTIMER_EdgeAlignedPwm

Edge-aligned PWM

enumerator kSCTIMER_CenterAlignedPwm

Center-aligned PWM

enum _sctimer_counter

SCTimer counters type.

Values:

enumerator kSCTIMER_Counter_L

16-bit Low counter.

enumerator kSCTIMER_Counter_H
16-bit High counter.

enumerator kSCTIMER_Counter_U
32-bit Unified counter.

enum _sctimer_input
List of SCTimer input pins.
Values:

enumerator kSCTIMER_Input_0
SCTIMER input 0

enumerator kSCTIMER_Input_1
SCTIMER input 1

enumerator kSCTIMER_Input_2
SCTIMER input 2

enumerator kSCTIMER_Input_3
SCTIMER input 3

enumerator kSCTIMER_Input_4
SCTIMER input 4

enumerator kSCTIMER_Input_5
SCTIMER input 5

enumerator kSCTIMER_Input_6
SCTIMER input 6

enumerator kSCTIMER_Input_7
SCTIMER input 7

enum _sctimer_out
List of SCTimer output pins.
Values:

enumerator kSCTIMER_Out_0
SCTIMER output 0

enumerator kSCTIMER_Out_1
SCTIMER output 1

enumerator kSCTIMER_Out_2
SCTIMER output 2

enumerator kSCTIMER_Out_3
SCTIMER output 3

enumerator kSCTIMER_Out_4
SCTIMER output 4

enumerator kSCTIMER_Out_5
SCTIMER output 5

enumerator kSCTIMER_Out_6
SCTIMER output 6

enumerator kSCTIMER_Out_7
SCTIMER output 7

enumerator kSCTIMER_Out_8
SCTIMER output 8

enumerator kSCTIMER_Out_9
SCTIMER output 9

enum _sctimer_pwm_level_select
SCTimer PWM output pulse mode: high-true, low-true or no output.

Values:

enumerator kSCTIMER_LowTrue
Low true pulses

enumerator kSCTIMER_HighTrue
High true pulses

enum _sctimer_pwm_update_mode
When an updated PWM duty cycle takes effect.

Values:

enumerator kSCTIMER_UpdateImmediately
Halt the counter, write the active MATCH register (and its reload), then restart: the new duty cycle takes effect at once. Writing the active MATCH register requires the counter halted.

enumerator kSCTIMER_UpdateOnNextPeriod
Write only the MATCH reload register while the counter keeps running: the new duty cycle takes effect at the next counter cycle (glitchless, no counter stall). Requires CONFIG[NORELOAD] clear for the affected counter.

enum _sctimer_clock_mode
SCTimer clock mode options.

Values:

enumerator kSCTIMER_System_ClockMode
System Clock Mode

enumerator kSCTIMER_Sampled_ClockMode
Sampled System Clock Mode

enumerator kSCTIMER_Input_ClockMode
SCT Input Clock Mode

enumerator kSCTIMER_Asynchronous_ClockMode
Asynchronous Mode

enum _sctimer_clock_select
SCTimer clock select options.
Values:

enumerator kSCTIMER_Clock_On_Rise_Input_0
Rising edges on input 0

enumerator kSCTIMER_Clock_On_Fall_Input_0
Falling edges on input 0

enumerator kSCTIMER_Clock_On_Rise_Input_1
Rising edges on input 1

enumerator kSCTIMER_Clock_On_Fall_Input_1
Falling edges on input 1

enumerator kSCTIMER_Clock_On_Rise_Input_2
Rising edges on input 2

enumerator kSCTIMER_Clock_On_Fall_Input_2
Falling edges on input 2

enumerator kSCTIMER_Clock_On_Rise_Input_3
Rising edges on input 3

enumerator kSCTIMER_Clock_On_Fall_Input_3
Falling edges on input 3

enumerator kSCTIMER_Clock_On_Rise_Input_4
Rising edges on input 4

enumerator kSCTIMER_Clock_On_Fall_Input_4
Falling edges on input 4

enumerator kSCTIMER_Clock_On_Rise_Input_5
Rising edges on input 5

enumerator kSCTIMER_Clock_On_Fall_Input_5
Falling edges on input 5

enumerator kSCTIMER_Clock_On_Rise_Input_6
Rising edges on input 6

enumerator kSCTIMER_Clock_On_Fall_Input_6
Falling edges on input 6

enumerator kSCTIMER_Clock_On_Rise_Input_7
Rising edges on input 7

enumerator kSCTIMER_Clock_On_Fall_Input_7
Falling edges on input 7

enum _sctimer_conflict_resolution
SCTimer output conflict resolution options.

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

Values:

enumerator kSCTIMER_ResolveNone
No change

enumerator kSCTIMER_ResolveSet
Set output

enumerator kSCTIMER_ResolveClear
Clear output

enumerator kSCTIMER_ResolveToggle
Toggle output

enum _sctimer_event_active_direction
List of SCTimer event generation active direction when the counters are operating in BIDIR mode.

Values:

enumerator kSCTIMER_ActiveIndependent
This event is triggered regardless of the count direction.

enumerator kSCTIMER_ActiveInCountUp
This event is triggered only during up-counting when BIDIR = 1.

enumerator kSCTIMER_ActiveInCountDown
This event is triggered only during down-counting when BIDIR = 1.

enum _sctimer_event

List of SCTimer event types.

Values:

enumerator kSCTIMER_InputLowOrMatchEvent

enumerator kSCTIMER_InputRiseOrMatchEvent

enumerator kSCTIMER_InputFallOrMatchEvent

enumerator kSCTIMER_InputHighOrMatchEvent

enumerator kSCTIMER_MatchEventOnly

enumerator kSCTIMER_InputLowEvent

enumerator kSCTIMER_InputRiseEvent

enumerator kSCTIMER_InputFallEvent

enumerator kSCTIMER_InputHighEvent

enumerator kSCTIMER_InputLowAndMatchEvent

enumerator kSCTIMER_InputRiseAndMatchEvent

enumerator kSCTIMER_InputFallAndMatchEvent

enumerator kSCTIMER_InputHighAndMatchEvent

enumerator kSCTIMER_OutputLowOrMatchEvent

enumerator kSCTIMER_OutputRiseOrMatchEvent

enumerator kSCTIMER_OutputFallOrMatchEvent

enumerator kSCTIMER_OutputHighOrMatchEvent

enumerator kSCTIMER_OutputLowEvent

enumerator kSCTIMER_OutputRiseEvent

enumerator kSCTIMER_OutputFallEvent

enumerator kSCTIMER_OutputHighEvent

enumerator kSCTIMER_OutputLowAndMatchEvent

enumerator kSCTIMER_OutputRiseAndMatchEvent

enumerator kSCTIMER_OutputFallAndMatchEvent

enumerator kSCTIMER_OutputHighAndMatchEvent

enum _sctimer_interrupt_enable

List of SCTimer interrupts.

Values:

enumerator kSCTIMER_Event0InterruptEnable

Event 0 interrupt

enumerator kSCTIMER_Event1InterruptEnable

Event 1 interrupt

enumerator kSCTIMER_Event2InterruptEnable

Event 2 interrupt

enumerator kSCTIMER_Event3InterruptEnable

Event 3 interrupt

enumerator kSCTIMER_Event4InterruptEnable

Event 4 interrupt

enumerator kSCTIMER_Event5InterruptEnable

Event 5 interrupt

enumerator kSCTIMER_Event6InterruptEnable

Event 6 interrupt

enumerator kSCTIMER_Event7InterruptEnable

Event 7 interrupt

enumerator kSCTIMER_Event8InterruptEnable

Event 8 interrupt

enumerator kSCTIMER_Event9InterruptEnable

Event 9 interrupt

enumerator kSCTIMER_Event10InterruptEnable

Event 10 interrupt

enumerator kSCTIMER_Event11InterruptEnable

Event 11 interrupt

enumerator kSCTIMER_Event12InterruptEnable
Event 12 interrupt

enum _sctimer_status_flags
List of SCTimer flags.
Values:

enumerator kSCTIMER_Event0Flag
Event 0 Flag

enumerator kSCTIMER_Event1Flag
Event 1 Flag

enumerator kSCTIMER_Event2Flag
Event 2 Flag

enumerator kSCTIMER_Event3Flag
Event 3 Flag

enumerator kSCTIMER_Event4Flag
Event 4 Flag

enumerator kSCTIMER_Event5Flag
Event 5 Flag

enumerator kSCTIMER_Event6Flag
Event 6 Flag

enumerator kSCTIMER_Event7Flag
Event 7 Flag

enumerator kSCTIMER_Event8Flag
Event 8 Flag

enumerator kSCTIMER_Event9Flag
Event 9 Flag

enumerator kSCTIMER_Event10Flag
Event 10 Flag

enumerator kSCTIMER_Event11Flag
Event 11 Flag

enumerator kSCTIMER_Event12Flag
Event 12 Flag

enumerator kSCTIMER_BusErrorLFlag
Bus error due to write when L counter was not halted

enumerator kSCTIMER_BusErrorHFlag

Bus error due to write when H counter was not halted

typedef enum *_sctimer_pwm_mode* sctimer_pwm_mode_t

SCTimer PWM operation modes.

typedef enum *_sctimer_counter* sctimer_counter_t

SCTimer counters type.

typedef enum *_sctimer_input* sctimer_input_t

List of SCTimer input pins.

typedef enum *_sctimer_out* sctimer_out_t

List of SCTimer output pins.

typedef enum *_sctimer_pwm_level_select* sctimer_pwm_level_select_t

SCTimer PWM output pulse mode: high-true, low-true or no output.

typedef struct *_sctimer_pwm_signal_param* sctimer_pwm_signal_param_t

Options to configure a SCTimer PWM signal.

typedef struct *_sctimer_complementary_pwm_handle* sctimer_complementary_pwm_handle_t

Handle for a complementary PWM output pair created by *SCTIMER_SetupComplementaryPwm()*.

The handle stores the event numbers and parameters needed to update the pair's duty cycle later with *SCTIMER_UpdateComplementaryPwmDutycycle()*. Dead time is inserted by delaying the relevant turn-on edge (the high side in edge-aligned mode, the low side in center-aligned mode) by deadTimeTicks, which guarantees a non-overlap gap at both switching edges.

typedef enum *_sctimer_pwm_update_mode* sctimer_pwm_update_mode_t

When an updated PWM duty cycle takes effect.

typedef enum *_sctimer_clock_mode* sctimer_clock_mode_t

SCTimer clock mode options.

typedef enum *_sctimer_clock_select* sctimer_clock_select_t

SCTimer clock select options.

typedef enum *_sctimer_conflict_resolution* sctimer_conflict_resolution_t

SCTimer output conflict resolution options.

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

typedef enum *_sctimer_event_active_direction* sctimer_event_active_direction_t

List of SCTimer event generation active direction when the counters are operating in BIDIR mode.

typedef enum *_sctimer_event* sctimer_event_t

List of SCTimer event types.

typedef void (*sctimer_event_callback_t)(void)

SCTimer callback typedef.

typedef enum *_sctimer_interrupt_enable* sctimer_interrupt_enable_t

List of SCTimer interrupts.

typedef enum *_sctimer_status_flags* sctimer_status_flags_t

List of SCTimer flags.

typedef struct *_sctimer_config* sctimer_config_t

SCTimer configuration structure.

This structure holds the configuration settings for the SCTimer peripheral. To initialize this structure to reasonable defaults, call the SCTMR_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

SCT_EV_STATE_STATEMSKn(x)

struct _sctimer_pwm_signal_param

#include <fsl_sctimer.h> Options to configure a SCTimer PWM signal.

Public Members

sctimer_out_t output

The output pin to use to generate the PWM signal

sctimer_pwm_level_select_t level

PWM output active level select.

uint8_t dutyCyclePercent

PWM pulse width, value should be between 0 to 100 0 = always inactive signal (0% duty cycle) 100 = always active signal (100% duty cycle).

struct _sctimer_complementary_pwm_handle

#include <fsl_sctimer.h> Handle for a complementary PWM output pair created by *SCTIMER_SetupComplementaryPwm()*.

The handle stores the event numbers and parameters needed to update the pair's duty cycle later with *SCTIMER_UpdateComplementaryPwmDutycycle()*. Dead time is inserted by delaying the relevant turn-on edge (the high side in edge-aligned mode, the low side in center-aligned mode) by deadTimeTicks, which guarantees a non-overlap gap at both switching edges.

Public Members

uint32_t periodEvent

Shared period/limit event; match = period. Set by frequency, constant across duty updates.

uint32_t highRiseEvent

Edge-aligned mode: the high-side turn-on event, match = deadTimeTicks (constant across duty updates). Center-aligned mode: there is no separate rise event, so this field is set equal to periodEvent and drives no output - do not treat it as a distinct event in that mode.

uint32_t highFallEvent

High-side turn-off event; match = pulse (= period * duty / 100). Rewritten on each duty update.

uint32_t lowRiseEvent

Low-side turn-on event; match = pulse + deadTimeTicks. Rewritten on each duty update.

uint32_t deadTimeTicks

Dead time in SCT counter clock ticks.

sctimer_out_t outHigh

High-side output pin.

sctimer_out_t outLow

Low-side output pin.

struct __sctimer__config

#include <fsl_sctimer.h> SCTimer configuration structure.

This structure holds the configuration settings for the SCTimer peripheral. To initialize this structure to reasonable defaults, call the SCTMR_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Public Members

bool enableCounterUnify

true: SCT operates as a unified 32-bit counter; false: SCT operates as two 16-bit counters. User can use the 16-bit low counter and the 16-bit high counters at the same time; for Hardware limit, user can not use unified 32-bit counter and any 16-bit low/high counter at the same time.

sctimer_clock_mode_t clockMode

SCT clock mode value

sctimer_clock_select_t clockSelect

SCT clock select value

bool enableBidirection_l

true: Up-down count mode for the L or unified counter false: Up count mode only for the L or unified counter

bool enableBidirection_h

true: Up-down count mode for the H or unified counter false: Up count mode only for the H or unified counter. This field is used only if the enableCounterUnify is set to false

uint8_t prescale_l

Prescale value to produce the L or unified counter clock

uint8_t prescale_h

Prescale value to produce the H counter clock. This field is used only if the enableCounterUnify is set to false

uint8_t outInitState

Defines the initial output value

uint8_t inputsync

SCT INSYNC value, INSYNC field in the CONFIG register, from bit9 to bit 16. it is used to define synchronization for input N: bit 9 = input 0 bit 10 = input 1 bit 11 = input 2 bit 12 = input 3 All other bits are reserved (bit13 ~bit 16). How User to set the the value for the member inputsync. IE: delay for input0, and input 1, bypasses for input 2 and input 3 MACRO definition in user level. #define INPUTSYNC0 (0U) #define INPUTSYNC1 (1U) #define INPUTSYNC2 (2U) #define INPUTSYNC3 (3U) User Code. sctimerInfo.inputsync = (1 « INPUTSYNC2) | (1 « INPUTSYNC3);

2.83 SDADC: SDADC Module

void SDADC_GetDefaultConfig(*sdadc_config_t* *config)

This function is used to get available predefined configurations for the SDADC initialization.

Parameters

- config – Pointer to the SDADC configuration structure, please refer to *sdadc_config_t* for details.

void SDADC_Init(SDADC_Type *base, const *sdadc_config_t* *config)

This function is used to initialize the SDADC.

Parameters

- base – SDADC peripheral base address.
- config – Pointer to the SDADC configuration structure, please refer to *sdadc_config_t* for details.

void SDADC_Deinit(SDADC_Type *base, const *sdadc_config_t* *config)

This function is used to de-initialize the SDADC.

Parameters

- base – SDADC peripheral base address.
- config – Pointer to the SDADC configuration structure, please refer to *sdadc_config_t* for details.

void SDADC_DoInitPowerUp(SDADC_Type *base, *sdadc_config_t* *config, uint32_t clock)

This function is used to power up the SDADC in the initialization state.

Parameters

- base – SDADC peripheral base address.
- config – Pointer to the SDADC configuration structure, please refer to *sdadc_config_t* for details.
- clock – Core clock frequency with Hz.

static inline void SDADC_ControlModulatorPowerEnable(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool enable)

This function is used to enable or disable SDADC specific channel's P-Side or/and N-Side sigma-delta modulator power.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.
- enable – Decides whether to enable or disable.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ControlReferencePowerEnable(SDADC_Type *base, bool enable)

This function is used to enable or disable the reference power.

Parameters

- base – SDADC peripheral base address.
- enable – Indicates enable or disable the reference power.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ConfigureReferenceMode(SDADC_Type *base, *sdadc_ref_mode_t* referenceMode)

This function is used to configure the reference mode.

Note: Setting the reference to fast charge mode is mainly used during startup where fast startup is required, the current consumption in this mode is the largest, so, it is recommended to switch to other modes to minimize current consumption. If the ADC works on the single-end mode, recommend switching to the low-noise mode, if the ADC works on the differential mode, recommend switching to the retain mode.

Parameters

- base – SDADC peripheral base address.
- referenceMode – Reference mode to be configured, see *sdadc_ref_mode_t* for details.

static inline void SDADC_ControlLdoPowerEnable(SDADC_Type *base, bool enable)

This function is used to enable or disable the LDO power.

Parameters

- base – SDADC peripheral base address.
- enable – Indicates enable or disable the LDO power.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ConfigureLdoDriveCapability(SDADC_Type *base, *sdadc_ldo_drive_capability_t* capability)

This function is used to configure the drive capability.

Parameters

- base – SDADC peripheral base address.
- capability – SDADC LDO drive capability to be set, see *sdadc_ldo_drive_capability_t* for details.

static inline void SDADC_ConfigureChannelMode(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_mode_t* mode)

This function is used to configure the channel mode.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- mode – channel mode, refer *sdadc_channel_mode_t* for details.

static inline void SDADC_ControlDacCompensationEnable(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool enable)

This function is used to enable or disable the DAC compensation function.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.
- enable – Decides whether to enable or disable.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ControlModulatorDcLoopEnable(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool enable)

This function is used to enable or disable the modulator DC Loop function.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.
- enable – Decides whether to enable or disable.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ControlDecimatorDcFilterEnable(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool enable)

This function is used to enable or disable the decimator DC Filter function.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.
- enable – Decides whether to enable or disable.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ControlDecimatorOutputInvertEnable(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool enable)

This function is used to enable or disable the decimator output invert.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.
- enable – Decides whether to enable or disable.
  - **true** Enable
  - **false** Disable

static inline void SDADC_ConfigureDecimatorVolume(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, *sdadc_volume_control_t* volume)

This function is used to configure the decimator volume.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.
- volume – Volume level, see *sdadc_volume_control_t* for details.

static inline void SDADC_ConfigureDecimatorSampleRate(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, *sdadc_sample_rate_t* sampleRate)

This function is used to configure the decimator sample rate.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.

- sampleRate – Sample rate, see *sdadc_sample_rate_t* for details.

static inline void SDADC_ConfigureDitherAmplitude(SDADC_Type *base, *sdadc_channel_number_t* number, uint8_t ditherAmplitude)

This function is used to set the dither amplitude.

In the ADC conversion process, adding dither can improve the ADC dynamic performance, but too large a dither will cause the input signal to overflow, the user needs to set the appropriate dither amplitude according to their own needs.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- ditherAmplitude – The dither amplitude that will be introduced, the range is from 000b to 111b, 000b means disable the dither.

static inline void SDADC_ControlModulatorReset(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool reset)

This function is used to reset or release reset for the SDADC specific channel's P-Side or N-Side sigma-delta modulator.

Note: Needs to be reset for at least 15us after start-up, and then release the reset.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.
- reset – Indicates reset or release reset for the sigma-delta modulator.
  - **true** Reset
  - **false** Release reset

static inline void SDADC_ControlDecimatorReset(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool reset)

This function is used to reset or release reset for the SDADC channeln's P-Side or/and N-Side decimator.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.
- reset – Indicates reset or release reset for the decimator.
  - **true** Reset
  - **false** Release reset

static inline void SDADC_ConfigureModulatorDcLoopReset(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, bool reset)

This function is used to reset or release reset for the SDADC channeln's P-Side or/and N-Side madulator DC Loop.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.
- reset – Indicates whether reset or release reset for the madulator DC Loop.
  - **true** Reset
  - **false** Release reset

static inline void SDADC_ControlGlobalFifoIntEnable(SDADC_Type *base, bool enable)

This function is used to enable or disable the global FIFO full interrupt.

Parameters

- base – SDADC peripheral base address.
- enable – Indicates whether enable or disable the global FIFO full interrupt.
  - **true** Enable interrupt
  - **false** Disable interrupt

static inline bool SDADC_CheckGlobalFifoInterrupted(SDADC_Type *base)

This function is used to check whether the FIFO full interrupt has occurred.

Parameters

- base – SDADC peripheral base address.

Returns

The FIFO full interrupt status flag.

- **true** The FIFO full interrupt has occurred.
- **false** The FIFO full interrupt has not occurred.

static inline void SDADC_ClearGlobalFifoIntStatusFlag(SDADC_Type *base)

This function is used to clear the FIFO full interrupt status flag.

Parameters

- base – SDADC peripheral base address.

static inline void SDADC_ControlFifoIntEnable(SDADC_Type *base, uint32_t mask, bool enable)

This function is used to enable or disable the FIFO full interrupt.

Parameters

- base – SDADC peripheral base address.
- mask – Mask value to enable or disable interrupt, please refer to *_sdadc_fifo_int_enable* for details.
- enable – Indicates enable or disable the FIFO full interrupt.
  - **true** Enable interrupt
  - **false** Disable interrupt

static inline void SDADC_ControlFifoWatermarkEnable(SDADC_Type *base, uint32_t mask, bool enable)

This function is used to enable or disable the FIFO watermark.

Parameters

- base – SDADC peripheral base address.

- mask – Mask value to enable or disable watermark, please refer to *_sdadc_fifo_watermark_enable* for details.
- enable – Indicates enable or disable the FIFO watermark.
  - **true** Enable watermark
  - **false** Disable watermark

static inline uint8_t SDADC_GetFifoIntStatusFlags(SDADC_Type *base)

This function is used to get the FIFO full interrupt status flags.

Parameters

- base – SDADC peripheral base address.

Returns

Status flags of the FIFO full interrupt.

static inline uint8_t SDADC_GetFifoWatermarkStatusFlags(SDADC_Type *base)

This function is used to get the FIFO watermark status flags.

Parameters

- base – SDADC peripheral base address.

Returns

Status flags of the FIFO watermark.

static inline void SDADC_ConfigureFifoWatermark(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, *sdadc_watermark_t* waterMark)

This function is used to configure FIFO watermark.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.
- waterMark – WaterMark value, see *sdadc_watermark_t* for details.

static inline void SDADC_ControlFifoReadWriteReset(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type, *sdadc_fifo_reset_operation_t* operation)

This function is used to reset the specific channel side FIFO's read or/and write pointer.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.
- operation – Indicates which reset operation to do, refer *sdadc_fifo_reset_operation_t* for details.

static inline uint8_t SDADC_GetConvChannelFifoAvailableEntries(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type)

This function is used to get the specific channel side FIFO's available entries.

Note: In the differential mode, the bit stream is converted to 24bit data (FIFO_x_P[23:0]) by the P-Side decimator, the output of N-Side decimator (FIFO_x_N[23:0]) is not valid.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.

Returns

The available entries of the specified FIFO, note that only the lower 5 bits of the returned data are valid.

static inline uint32_t SDADC_GetConvChannelFifoRawData(SDADC_Type *base, *sdadc_channel_number_t* number, *sdadc_channel_type_t* type)

This function is used to get the specific channel side FIFO's raw data.

Note: In the differential mode, the bit stream is converted to 24bit data (FIFO_x_P[23:0]) by the P-Side decimator, the output of N-Side decimator (FIFO_x_N[23:0]) is not valid. So the type that needs to be passed in in differential mode is *kSDADC_Channel_PSide_Type*.

Parameters

- base – SDADC peripheral base address.
- number – channel number, refer *sdadc_channel_number_t* for details.
- type – channel type, refer *sdadc_channel_type_t* for details.

Returns

The raw data of the specified FIFO.

void SDADC_CopyConvChannelFifoToBuffer(SDADC_Type *base, *sdadc_channel_group* *group, uint8_t count, void *buffer)

This function is used to copy specific conversion channels' data from FIFO to buffer.

Note: This function will copy all 16 entries in the specified FIFO.

Parameters

- base – SDADC peripheral base address.
- group – Indicates which group of channels data to obtain.
- count – Indicates how many channels in the specified group.
- buffer – The buffer which stores conversion data.

FSL_SDADC_DRIVER_VERSION

SDADC driver version 2.0.1.

enum _sdadc_fifo_int_enable

This enumeration provides the mask for the SDADC channel P-Side and N-Side FIFO full interrupt enable/disable.

Values:

enumerator kSDADC_Channel0_PSide_FifoFullIntEnable

enumerator kSDADC_Channel0_NSide_FifoFullIntEnable

enumerator kSDADC_Channel1_PSide_FifoFullIntEnable

enumerator kSDADC_Channel1_NSide_FifoFullIntEnable

enumerator kSDADC_Channel2_PSide_FifoFullIntEnable

enumerator kSDADC_Channel2_NSide_FifoFullIntEnable

enumerator kSDADC_Channel3_PSide_FifoFullIntEnable

enumerator kSDADC_Channel3_NSide_FifoFullIntEnable

enum _sdadc_fifo_watermark_enable

This enumeration provides the mask for the SDADC channel P-Side and N-Side FIFO watermark enable/disable.

Values:

enumerator kSDADC_Channel0_PSide_FifoWatermarkEnable

enumerator kSDADC_Channel0_NSide_FifoWatermarkEnable

enumerator kSDADC_Channel1_PSide_FifoWatermarkEnable

enumerator kSDADC_Channel1_NSide_FifoWatermarkEnable

enumerator kSDADC_Channel2_PSide_FifoWatermarkEnable

enumerator kSDADC_Channel2_NSide_FifoWatermarkEnable

enumerator kSDADC_Channel3_PSide_FifoWatermarkEnable

enumerator kSDADC_Channel3_NSide_FifoWatermarkEnable

enum _sdadc_fifo_int_status_flag

This enumeration provides the mask for the SDADC channel P-Side and N-Side FIFO full interrupt status flag.

Values:

enumerator kSDADC_Channel0_PSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel0_NSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel1_PSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel1_NSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel2_PSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel2_NSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel3_PSide_FifoFullIntStatusFlag

enumerator kSDADC_Channel3_NSide_FifoFullIntStatusFlag

enum _sdadc_fifo_watermark_status_flag

This enumeration provides the mask for the SDADC channel P-Side and N-Side FIFO watermark status flag.

Values:

enumerator kSDADC_Channel0_PSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel0_NSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel1_PSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel1_NSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel2_PSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel2_NSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel3_PSide_FifoWatermarkStatusFlag

enumerator kSDADC_Channel3_NSide_FifoWatermarkStatusFlag

enum _sdadc_channel_mode

This enumeration provides the selection of the SDADC channel mode.

Values:

enumerator kSDADC_SingleEnd_Mode

Single-end mode.

enumerator kSDADC_Differential_Mode

Differential mode.

enum _sdadc_channel_number

This enumeration provides the selection of the SDADC channel number.

Values:

enumerator kSDADC_Channel0

SDADC channel 0.

enumerator kSDADC_Channel1

SDADC channel 1.

enumerator kSDADC_Channel2

SDADC channel 2.

enumerator kSDADC_Channel3

SDADC channel 3.

enum _sdadc_channel_type

This enumeration provides the selection of the SDADC channel type.

Values:

enumerator kSDADC_Channel_NSide_Type

N-side selected.

enumerator kSDADC_Channel_PSide_Type

P-side selected.

enumerator kSDADC_Channel_NandPSide_Type

N- and P-side selected.

enum _sdadc_decimator_sample_rate

This enumeration provides the selection of the SDADC decimator sample rate.

Values:

enumerator kSDADC_DecimatorSampleRate48kHz

48kHz decimator sample rate.

enumerator kSDADC_DecimatorSampleRate32kHz

32kHz decimator sample rate.

enumerator kSDADC_DecimatorSampleRate16kHz

16kHz decimator sample rate.

enum _sdadc_ldo_drive_capability

This enumeration provides the selection of the SDADC LDO drive capability.

Values:

enumerator kSDADC_LdoLowPower

To drive 1 single-end mode or 1 differential mode SDADC.

enumerator kSDADC_LdoHighPower

To drive more than 1 SDADC

enum _sdadc_ref_mode

This enumeration provides the selection of the SDADC reference mode.

The SDADC reference module has four modes, these modes are controlled via REF_CTL register 'fast_ref_enable' and 'pwr_fast_ref_enable' bitfields. 1.Retain mode: pwr_fast_ref_enable = 0b, fast_ref_enable = 0b. 2.Low noise mode: pwr_fast_ref_enable = 0b, fast_ref_enable = 1b. 3.Low power mode: pwr_fast_ref_enable = 1b, fast_ref_enable = 0b. 4.Fast charge mode: pwr_fast_ref_enable = 1b, fast_ref_enable = 1b.

Values:

enumerator kSDADC_RefRetainMode

Reference in retain mode.

enumerator kSDADC_RefLowNoiseMode

Reference in low noise mode.

enumerator kSDADC_RefLowPowerMode

Reference in low power mode.

enumerator kSDADC_RefFastChargeMode

Reference in fast charge mode.

enum _sdadc_fifo_reset_operation

This enumeration provides the selection of the SDADC FIFO write or/and read operation.

Values:

enumerator kSDADC_FifoWriteReset

FIFO write reset.

enumerator kSDADC_FifoReadReset

FIFO read reset.

enumerator kSDADC_FifoWriteAndReadReset

FIFO write and read reset.

typedef enum *_sdadc_channel_mode* sdadc_channel_mode_t

This enumeration provides the selection of the SDADC channel mode.

typedef enum *_sdadc_channel_number* sdadc_channel_number_t

This enumeration provides the selection of the SDADC channel number.

typedef enum *_sdadc_channel_type* sdadc_channel_type_t

This enumeration provides the selection of the SDADC channel type.

typedef enum *_sdadc_decimator_sample_rate* sdadc_decimator_sample_rate_t

This enumeration provides the selection of the SDADC decimator sample rate.

typedef enum *_sdadc_ldo_drive_capability* sdadc_ldo_drive_capability_t

This enumeration provides the selection of the SDADC LDO drive capability.

typedef enum *_sdadc_ref_mode* sdadc_ref_mode_t

This enumeration provides the selection of the SDADC reference mode.

The SDADC reference module has four modes, these modes are controlled via REF_CTL register 'fast_ref_enable' and 'pwr_fast_ref_enable' bitfields. 1.Retain mode: pwr_fast_ref_enable = 0b, fast_ref_enable = 0b. 2.Low noise mode: pwr_fast_ref_enable = 0b, fast_ref_enable = 1b. 3.Low power mode: pwr_fast_ref_enable = 1b, fast_ref_enable = 0b. 4.Fast charge mode: pwr_fast_ref_enable = 1b, fast_ref_enable = 1b.

typedef enum *_sdadc_fifo_reset_operation* sdadc_fifo_reset_operation_t

This enumeration provides the selection of the SDADC FIFO write or/and read operation.

typedef struct *_sdadc_volume* sdadc_volume_control_t

This structure is used to configure the decimator volume (gain).

Note: The gain setting is not applicable for the channel N-Side when executing the differential mode. For the specific gain setting value please refer to the reference manual.

---

typedef struct *_sdadc_sample_rate* sdadc_sample_rate_t

This structure is used to configure the decimator sample rate.

typedef struct *_sdadc_watermark* sdadc_watermark_t

This structure is used to configure the FIFO watermark.

Note: In the differential mode, the bitstream is converted to 24bit data (FIFO_x_P[23:0]) by P-Side decimator, the output of N-Side decimator (FIFO_x_N[23:0]) is not valid.

---

typedef struct *_sdadc_channel_config* sdadc_channel_config_t

This structure is used to configure the SDADC channels.

typedef struct *_sdadc_config* sdadc_config_t

This structure is used to configure the SDADC module.

typedef struct *_sdadc_channel_group* sdadc_channel_group

This structure is used to construct channel groups.

SDADC_DECIMATOR_VOL_CTRL_P_MASK(offset)

Macros are used to configure decimator volume.

SDADC_DECIMATOR_VOL_CTRL_P_SHIFT(offset)

SDADC_DECIMATOR_VOL_CTRL_P(volume, offset)

SDADC_DECIMATOR_VOL_CTRL_N_MASK(offset)

SDADC_DECIMATOR_VOL_CTRL_N_SHIFT(offset)

SDADC_DECIMATOR_VOL_CTRL_N(volume, offset)

SDADC_DECIMATOR_SET_SPEED_P_MASK(offset)
Macros are used to configure decimator sample rate.

SDADC_DECIMATOR_SET_SPEED_P_SHIFT(offset)

SDADC_DECIMATOR_SET_SPEED_P(rate, offset)

SDADC_DECIMATOR_SET_SPEED_N_MASK(offset)

SDADC_DECIMATOR_SET_SPEED_N_SHIFT(offset)

SDADC_DECIMATOR_SET_SPEED_N(rate, offset)

SDADC_ADC_CTL_2_SET_DITHER_MASK(offset)
Macros are used to configure the dither amplitude.

SDADC_ADC_CTL_2_SET_DITHER_SHIFT(offset)

SDADC_ADC_CTL_2_SET_DITHER(val, offset)

SDADC_FIFO_WATERMARK_CTL_P_MASK(offset)
Macros are used to configure FIFO watermark.

SDADC_FIFO_WATERMARK_CTL_P_SHIFT(offset)

SDADC_FIFO_WATERMARK_CTL_P(watermark, offset)

SDADC_FIFO_WATERMARK_CTL_N_MASK(offset)

SDADC_FIFO_WATERMARK_CTL_N_SHIFT(offset)

SDADC_FIFO_WATERMARK_CTL_N(watermark, offset)

SDADC_FIFO_WATERMARK_STATUS_MASK
Macros are used for FIFO operations.

SDADC_FIFO_WATERMARK_STATUS_SHIFT

SDADC_FIFO_FULL_INT_STATUS_MASK

SDADC_FIFO_FULL_INT_STATUS_SHIFT

SDADC_FIFO_MASK

SDADC_FIFO_SHIFT

SDADC_FIFO_ENTRIES_AVAIL_MASK

SDADC_FIFO_DEPTH

SDADC_FIFO_WIDTH

struct _sdadc_volume
#include <fsl_sdadc.h> This structure is used to configure the decimator volume (gain).

Note: The gain setting is not applicable for the channel N-Side when executing the differential mode. For the specific gain setting value please refer to the reference manual.

Public Members

uint8_t pSideVolume
: Sets the gain for the P-Side decimator.

uint8_t nSideVolume
: Sets the gain for the N-Side decimator.

struct _sdadc_sample_rate
: *#include <fsl_sdadc.h>* This structure is used to configure the decimator sample rate.

Public Members

sdadc_decimator_sample_rate_t pSideSampleRate
: Sets the P-Side decimator sample rate.

sdadc_decimator_sample_rate_t nSideSampleRate
: Sets the N-Side decimator sample rate.

struct _sdadc_watermark
: *#include <fsl_sdadc.h>* This structure is used to configure the FIFO watermark.

Note: In the differential mode, the bitstream is converted to 24bit data (FIFO_x_P[23:0]) by P-Side decimator, the output of N-Side decimator (FIFO_x_N[23:0]) is not valid.

Public Members

uint8_t pSideWatermark
: Sets the channel N-Side FIFO watermark.

uint8_t nSideWatermark
: Sets the channel P-Side FIFO watermark.

struct _sdadc_channel_config
: *#include <fsl_sdadc.h>* This structure is used to configure the SDADC channels.

Public Members

sdadc_channel_mode_t mode
: Sets channel mode.

sdadc_channel_number_t number
: Sets channel number.

sdadc_channel_type_t type

Sets channel type.

sdadc_volume_control_t volume

Sets N- and P-side decimator volume.

sdadc_sample_rate_t samplerate

Sets N- and P-side decimator samplerate.

sdadc_watermark_t watermark

Sets N- and P-side FIFO watermark.

bool enableDacCompensation

Decides whether to enable the compensation feature for the ADCp[3:0] or/and ADCn[3:0] in the single-end mode, is used to avoid large data-dependent ripple on LDO output which will impact the ADC performance.

bool enableDCLoop

In some cases of the single-end mode, the microphone output is connected directly to ADC input without the coupling capacitor, DC Loop is activated to regulate input common-mode voltage of the ADCp[3:0] or/and ADCn[3:0]. In the single-end mode, there is the possibility to connect MEMS or JFET microphone directly to the input (DC-coupled). In this configuration, DC Loop mode needs to be activated to establish the correct common mode input voltage level at the input pin. The DC Loop function should disabled in the differential mode.

bool enableDcFilter

Decides whether to enable the decimator DC Filter to remove DC components.

bool enablePolarityInvert

Decides whether to enable the polarity invert function, for example: if the input to ADC is a pure sinus signal, then if the polarity invert is enabled, the decimation output will be a sinus signal 180 out of phase with the input.

struct _sdadc_config

#include <fsl_sdadc.h> This structure is used to configure the SDADC module.

Public Members

uint8_t channelCount

Indicates how many channels to execute the conversion.

sdadc_channel_config_t *channelConfig

SDADC channels configuration.

struct _sdadc_channel_group

#include <fsl_sdadc.h> This structure is used to construct channel groups.

Public Members

sdadc_channel_number_t number
Channel number.

sdadc_channel_type_t type
Channel type.

2.84 SEMA42: Hardware Semaphores Driver

FSL_SEMA42_DRIVER_VERSION
SEMA42 driver version.

SEMA42 status return codes.

Values:

enumerator kStatus_SEMA42_Busy
SEMA42 gate has been locked by other processor.

enumerator kStatus_SEMA42_Reseting
SEMA42 gate reseting is ongoing.

enum _sema42_gate_status
SEMA42 gate lock status.

Values:

enumerator kSEMA42_Unlocked
The gate is unlocked.

enumerator kSEMA42_LockedByProc0
The gate is locked by processor 0.

enumerator kSEMA42_LockedByProc1
The gate is locked by processor 1.

enumerator kSEMA42_LockedByProc2
The gate is locked by processor 2.

enumerator kSEMA42_LockedByProc3
The gate is locked by processor 3.

enumerator kSEMA42_LockedByProc4
The gate is locked by processor 4.

enumerator kSEMA42_LockedByProc5
The gate is locked by processor 5.

enumerator kSEMA42_LockedByProc6
The gate is locked by processor 6.

enumerator kSEMA42_LockedByProc7
The gate is locked by processor 7.

enumerator kSEMA42_LockedByProc8
The gate is locked by processor 8.

enumerator kSEMA42_LockedByProc9
The gate is locked by processor 9.

enumerator kSEMA42_LockedByProc10
The gate is locked by processor 10.

enumerator kSEMA42_LockedByProc11
The gate is locked by processor 11.

enumerator kSEMA42_LockedByProc12
The gate is locked by processor 12.

enumerator kSEMA42_LockedByProc13
The gate is locked by processor 13.

enumerator kSEMA42_LockedByProc14
The gate is locked by processor 14.

typedef enum *_sema42_gate_status* sema42_gate_status_t
SEMA42 gate lock status.

void SEMA42_Init(SEMA42_Type *base)
Initializes the SEMA42 module.

This function initializes the SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA42_ResetGate or SEMA42_ResetAllGates function.

Parameters

- base – SEMA42 peripheral base address.

void SEMA42_Deinit(SEMA42_Type *base)
De-initializes the SEMA42 module.

This function de-initializes the SEMA42 module. It only disables the clock.

Parameters

- base – SEMA42 peripheral base address.

status_t SEMA42_TryLock(SEMA42_Type *base, uint8_t gateNum, uint8_t procNum)

Tries to lock the SEMA42 gate.

This function tries to lock the specific SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – Lock the sema42 gate successfully.
- kStatus_SEMA42_Busy – Sema42 gate has been locked by another processor.

status_t SEMA42_Lock(SEMA42_Type *base, uint8_t gateNum, uint8_t procNum)

Locks the SEMA42 gate.

This function locks the specific SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

If SEMA42_BUSY_POLL_COUNT is defined and non-zero, the function will timeout after the specified number of polling iterations and return kStatus_Timeout.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to lock.
- procNum – Current processor number.

Return values

- kStatus_Success – The gate was successfully locked.
- kStatus_Timeout – Timeout occurred while waiting for the gate to be unlocked.

Returns

status_t

static inline void SEMA42_Unlock(SEMA42_Type *base, uint8_t gateNum)

Unlocks the SEMA42 gate.

This function unlocks the specific SEMA42 gate. It only writes unlock value to the SEMA42 gate register. However, it does not check whether the SEMA42 gate is locked by the current processor or not. As a result, if the SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number to unlock.

static inline *sema42_gate_status_t* SEMA42_GetGateStatus(SEMA42_Type *base, uint8_t gateNum)

Gets the status of the SEMA42 gate.

This function checks the lock status of a specific SEMA42 gate.

Parameters

- base – SEMA42 peripheral base address.

- gateNum – Gate number.

Returns
status Current status.

status_t SEMA42_ResetGate(SEMA42_Type *base, uint8_t gateNum)

Resets the SEMA42 gate to an unlocked status.

This function resets a SEMA42 gate to an unlocked status.

Parameters

- base – SEMA42 peripheral base address.
- gateNum – Gate number.

Return values

- kStatus_Success – SEMA42 gate is reset successfully.
- kStatus_SEMA42_Reseting – Some other reset process is ongoing.

static inline *status_t* SEMA42_ResetAllGates(SEMA42_Type *base)

Resets all SEMA42 gates to an unlocked status.

This function resets all SEMA42 gate to an unlocked status.

Parameters

- base – SEMA42 peripheral base address.

Return values

- kStatus_Success – SEMA42 is reset successfully.
- kStatus_SEMA42_Reseting – Some other reset process is ongoing.

SEMA42_GATE_NUM_RESET_ALL

The number to reset all SEMA42 gates.

SEMA42_GATEn(base, n)

SEMA42 gate n register address.

The SEMA42 gates are sorted in the order 3, 2, 1, 0, 7, 6, 5, 4, ... not in the order 0, 1, 2, 3, 4, 5, 6, 7, ... The macro SEMA42_GATEn gets the SEMA42 gate based on the gate index.

The input gate index is XOR'ed with 3U: 0 ^ 3 = 3 1 ^ 3 = 2 2 ^ 3 = 1 3 ^ 3 = 0 4 ^ 3 = 7 5 ^ 3 = 6 6 ^ 3 = 5 7 ^ 3 = 4 ...

SEMA42_BUSY_POLL_COUNT

Maximum polling iterations for SEMA42 waiting loops.

This parameter defines the maximum number of iterations for any polling loop in the SEMA42 driver code before timing out and returning an error.

It applies to all waiting loops in SEMA42 driver, such as waiting for a gate to be unlocked, waiting for a reset to complete, or waiting for a resource to become available.

This is a count of loop iterations, not a time-based value.

If defined as 0, polling loops will continue indefinitely until their exit condition is met, which could potentially cause the system to hang if hardware doesn't respond or if a resource is never released.

2.85 Soc_mipi_dsi

FSL_SOC_MIPI_DSI_DRIVER_VERSION
Driver version.

DSI_DPHY_PLL_VCO_MAX

DSI_DPHY_PLL_VCO_MIN

FSL_COMPONENT_ID

2.86 SYSPM: System Performance Monitor

enum _syspm_monitor
syspm select control monitor
Values:

enumerator kSYSPM_Monitor0
Monitor 0

enum _syspm_event
syspm select event
Values:

enumerator kSYSPM_Event1
Event 1

enumerator kSYSPM_Event2
Event 2

enumerator kSYSPM_Event3
Event 3

enum _syspm_mode
syspm set count mode
Values:

enumerator kSYSPM_BothMode
count in both modes

enumerator kSYSPM_UserMode
count only in user mode

enumerator kSYSPM_PrivilegedMode
count only in privileged mode

enum _syspm_startstop_control

syspm start/stop control

Values:

enumerator kSYSPM_Idle

idle >

enumerator kSYSPM_LocalStop

local stop

enumerator kSYSPM_LocalStart

local start

enumerator KSYSPM_EnableTraceControl

enable global TSTART/TSTOP

enumerator kSYSPM_GlobalStart

global stop

enumerator kSYSPM_GlobalStop

global start

typedef enum *_syspm_monitor* syspm_monitor_t

syspm select control monitor

typedef enum *_syspm_event* syspm_event_t

syspm select event

typedef enum *_syspm_mode* syspm_mode_t

syspm set count mode

typedef enum *_syspm_startstop_control* syspm_startstop_control_t

syspm start/stop control

void SYSPM_Init(SYSPM_Type *base)

Initializes the SYSPM.

This function enables the SYSPM clock.

Parameters

- base – SYSPM peripheral base address.

void SYSPM_Deinit(SYSPM_Type *base)

Deinitializes the SYSPM.

This function disables the SYSPM clock.

Parameters

- base – SYSPM peripheral base address.

void SYSPM_SelectEvent(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_event_t* event, uint8_t eventCode)

Select event counters.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.
- event – syspm select event, see to *syspm_event_t*.
- eventCode – select which event to be counted in PMECTRx., see to table Events.

void SYSPM_ResetEvent(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_event_t* event)

Reset event counters.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.
- event – syspm select event, see to *syspm_event_t*.

void SYSPM_ResetInstructionEvent(SYSPM_Type *base, *syspm_monitor_t* monitor)

Reset Instruction Counter.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.

void SYSPM_SetCountMode(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_mode_t* mode)

Set count mode.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.
- mode – syspm select counter mode, see to *syspm_mode_t*.

void SYSPM_SetStartStopControl(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_startstop_control_t* ssc)

Set Start/Stop Control.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.
- ssc – This 3-bit field provides a three-phase mechanism to start/stop the counters. It includes a prioritized scheme with local start > local stop > global start > global stop > conditional TSTART > TSTOP. The global and conditional start/stop affect all configured PM/PSAM module concurrently so counters are “coherent”. see to *syspm_startstop_control_t*

void SYSPM_DisableCounter(SYSPM_Type *base, *syspm_monitor_t* monitor)

Disable Counters if Stopped or Halted.

Parameters

- base – SYSPM peripheral base address.

- monitor – syspm control monitor, see to *syspm_monitor_t*.

uint64_t SYSPM_GetEventCounter(SYSPM_Type *base, *syspm_monitor_t* monitor, *syspm_event_t* event)

This is the the 40-bits of eventx counter. The value in this register increments each time the event selected in PMCRx[SELEVTx] occurs.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.
- event – syspm select event, see to *syspm_event_t*.

Returns

- When the return value is not equal to *SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE*, the return value represents a 40 bits eventx counter.
- When the return value is equal to *SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE*, the return value represents timeout occured.

uint64_t SYSPM_GetInstructionCounter(SYSPM_Type *base, *syspm_monitor_t* monitor)

This is the the 40-bits of instructionx counter. The value in this register increments each time the CPU count signals occurs.

Parameters

- base – SYSPM peripheral base address.
- monitor – syspm control monitor, see to *syspm_monitor_t*.

Returns

- When the return value is not equal to *SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE*, the return value represents a 40 bits instruction counter.
- When the return value is equal to *SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE*, the return value represents timeout occured.

EVENT_COUNT_STABLE_TIMEOUT

Max loops to wait for SYSPM event count stable (0 means wait forever)

INSTRUCTION_COUNT_STABLE_TIMEOUT

Max loops to wait for SYSPM instruction count stable (0 means wait forever)

FSL_SYSPM_DRIVER_VERSION

SYSPM driver version.

SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE

2.87 TRNG: True Random Number Generator

FSL_TRNG_DRIVER_VERSION

TRNG driver version 2.0.23.

Current version: 2.0.23

Change log:

- version 2.0.23
  - Updated TRNG default values for MCXA577 devices based on silicon characterization.
- version 2.0.22
  - Added support for KW43 and MCXW70 devices.
- version 2.0.21
  - Added support for MCXC devices.
- version 2.0.20
  - Added support for MCXA devices.
- version 2.0.19
  - Added support for MCXA and MCXL.
- version 2.0.18
  - TRNG health checks now done in software on RT5xx and RT6xx.
- version 2.0.17
  - Added support for RT700.
- version 2.0.16
  - Added support for Dual oscillator mode.
- version 2.0.15
  - Changed TRNG_USER_CONFIG_DEFAULT_XXX values according to latest reccomended by design team.
- version 2.0.14
  - add support for RW610 and RW612
- version 2.0.13
  - After deepsleep it might return error, added clearing bits in *TRNG_GetRandomData()* and generating new entropy.
  - Modified reloading entropy in *TRNG_GetRandomData()*, for some data length it doesn't reloading entropy correctly.
- version 2.0.12
  - For KW34A4_SERIES, KW35A4_SERIES, KW36A4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.
- version 2.0.11
  - Add clearing pending errors in *TRNG_Init()*.
- version 2.0.10
  - Fixed doxygen issues.
- version 2.0.9
  - Fix HIS_CCM metrics issues.
- version 2.0.8
  - For K32L2A41A_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv4.
- version 2.0.7

  - Fix MISRA 2004 issue rule 12.5.
- version 2.0.6
  - For KW35Z4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.
- version 2.0.5
  - Add possibility to define default TRNG configuration by device specific preprocessor macros for FRQMIN, FRQMAX and OSCDIV.
- version 2.0.4
  - Fix MISRA-2012 issues.
- Version 2.0.3
  - update TRNG_Init to restart entropy generation
- Version 2.0.2
  - fix MISRA issues
- Version 2.0.1
  - add support for KL8x and KL28Z
  - update default OSCDIV for K81 to divide by 2

enum _trng_sample_mode

TRNG sample mode. Used by trng_config_t.

Values:

enumerator kTRNG_SampleModeVonNeumann

Use von Neumann data in both Entropy shifter and Statistical Checker.

enumerator kTRNG_SampleModeRaw

Use raw data into both Entropy shifter and Statistical Checker.

enumerator kTRNG_SampleModeVonNeumannRaw

Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

enum _trng_clock_mode

TRNG clock mode. Used by trng_config_t.

Values:

enumerator kTRNG_ClockModeRingOscillator

Ring oscillator is used to operate the TRNG (default).

enumerator kTRNG_ClockModeSystem

System clock is used to operate the TRNG. This is for test use only, and indeterminate results may occur.

enum _trng_ring_osc_div

TRNG ring oscillator divide. Used by trng_config_t.

Values:

enumerator kTRNG_RingOscDiv0
Ring oscillator with no divide

enumerator kTRNG_RingOscDiv2
Ring oscillator divided-by-2.

enumerator kTRNG_RingOscDiv4
Ring oscillator divided-by-4.

enumerator kTRNG_RingOscDiv8
Ring oscillator divided-by-8.

enum trng_oscillator_mode_t
TRNG oscillator mode . Used by trng_config_t.

Values:

enumerator kTRNG_SingleOscillatorModeOsc1
Single oscillator mode, using OSC1 (default)

enumerator kTRNG_DualOscillatorMode
Dual oscillator mode

enumerator kTRNG_SingleOscillatorModeOsc2
Single oscillator mode, using OSC2

typedef enum *_trng_sample_mode* trng_sample_mode_t
TRNG sample mode. Used by trng_config_t.

typedef enum *_trng_clock_mode* trng_clock_mode_t
TRNG clock mode. Used by trng_config_t.

typedef enum *_trng_ring_osc_div* trng_ring_osc_div_t
TRNG ring oscillator divide. Used by trng_config_t.

typedef enum *trng_oscillator_mode_t* trng_oscillator_mode_t
TRNG oscillator mode . Used by trng_config_t.

typedef struct *_trng_statistical_check_limit* trng_statistical_check_limit_t
Data structure for definition of statistical check limits. Used by trng_config_t.

typedef struct *_trng_user_config* trng_config_t
Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the *TRNG_Init()* function. It contains all TRNG configurations.

status_t TRNG_GetDefaultConfig(*trng_config_t* *userConfig)
Initializes the user configuration structure to default values.

This function initializes the configuration structure to default values. The default values are platform dependent.

Parameters

- userConfig – User configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

status_t TRNG_Init(TRNG_Type *base, const *trng_config_t* *userConfig)

Initializes the TRNG.

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

Parameters

- base – TRNG base address
- userConfig – Pointer to the initialization configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

void TRNG_Deinit(TRNG_Type *base)

Shuts down the TRNG.

This function shuts down the TRNG.

Parameters

- base – TRNG base address.

status_t TRNG_GetRandomData(TRNG_Type *base, void *data, size_t dataSize)

Gets random data.

This function gets random data from the TRNG.

Parameters

- base – TRNG base address.
- data – Pointer address used to store random data.
- dataSize – Size of the buffer pointed by the data parameter.

Returns

random data

struct __trng_statistical_check_limit

#include <fsl_trng.h> Data structure for definition of statistical check limits. Used by trng_config_t.

Public Members

uint32_t maximum

Maximum limit.

int32_t minimum

Minimum limit.

struct __trng__user__config

#include <fsl_trng.h> Data structure for the TRNG initialization.

This structure initializes the TRNG by calling the *TRNG_Init()* function. It contains all TRNG configurations.

Public Members

bool lock

Disable programmability of TRNG registers.

trng_clock_mode_t clockMode

Clock mode used to operate TRNG.

trng_ring_osc_div_t ringOscDiv

Ring oscillator divide used by TRNG.

trng_sample_mode_t sampleMode

Sample mode of the TRNG ring oscillator.

trng_oscillator_mode_t oscillatorMode

TRNG oscillator mode .

trng_ring_osc_div_t ringOsc2Div

Divider used for Ring oscillator 2.

uint16_t entropyDelay

Entropy Delay. Defines the length (in system clocks) of each Entropy sample taken.

uint16_t sampleSize

Sample Size. Defines the total number of Entropy samples that will be taken during Entropy generation.

uint16_t sparseBitLimit

Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated. This limit is used only for during von Neumann sampling (enabled by TRNG_HAL_SetSampleMode()). Samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy.

uint8_t retryCount

Retry count. It defines the number of times a statistical check may fails during the TRNG Entropy Generation before generating an error.

uint8_t longRunMaxLimit

Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.

trng_statistical_check_limit_t monobitLimit

Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.

trng_statistical_check_limit_t runBit1Limit

Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.

trng_statistical_check_limit_t runBit2Limit

Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.

trng_statistical_check_limit_t runBit3Limit

Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.

trng_statistical_check_limit_t runBit4Limit

Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.

trng_statistical_check_limit_t runBit5Limit

Maximum and minimum limits for statistical check of number of runs of length 5 detected during entropy generation.

trng_statistical_check_limit_t runBit6PlusLimit

Maximum and minimum limits for statistical check of number of runs of length 6 or more detected during entropy generation.

trng_statistical_check_limit_t pokerLimit

Maximum and minimum limits for statistical check of "Poker Test".

trng_statistical_check_limit_t frequencyCountLimit

Maximum and minimum limits for statistical check of entropy sample frequency count.

2.88 USDHC: Ultra Secured Digital Host Controller Driver

void USDHC_Init(USDHC_Type *base, const *usdhc_config_t* *config)

USDHC module initialization function.

Configures the USDHC according to the user configuration.

Example:

```
usdhc_config_t config;
config.cardDetectDat3 = false;
config.endianMode = kUSDHC_EndianModeLittle;
config.dmaMode = kUSDHC_DmaModeAdma2;
config.readWatermarkLevel = 128U;
config.writeWatermarkLevel = 128U;
USDHC_Init(USDHC, &config);
```

Parameters

- base – USDHC peripheral base address.
- config – USDHC configuration information.

Return values

kStatus_Success – Operate successfully.

void USDHC_Deinit(USDHC_Type *base)

Deinitializes the USDHC.

Parameters

- base – USDHC peripheral base address.

bool USDHC_Reset(USDHC_Type *base, uint32_t mask, uint32_t timeout)

Resets the USDHC.

Parameters

- base – USDHC peripheral base address.
- mask – The reset type mask(*_usdhc_reset*).
- timeout – Timeout for reset.

Return values

- true – Reset successfully.
- false – Reset failed.

status_t USDHC_SetAdmaTableConfig(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, *usdhc_data_t* *dataConfig, uint32_t flags)

Sets the DMA descriptor table configuration. A high level DMA descriptor configuration function.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – ADMA configuration
- dataConfig – Data descriptor
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum *_usdhc_adma_flag*.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

status_t USDHC_SetInternalDmaConfig(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, const uint32_t *dataAddr, bool enAutoCmd23)

Internal DMA configuration. This function is used to config the USDHC DMA related registers.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – ADMA configuration.
- dataAddr – Transfer data address, a simple DMA parameter, if ADMA is used, leave it to NULL.

- enAutoCmd23 – Flag to indicate Auto CMD23 is enable or not, a simple DMA parameter, if ADMA is used, leave it to false.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

status_t USDHC_SetADMA2Descriptor(uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)

Sets the ADMA2 descriptor table configuration.

Parameters

- admaTable – ADMA table address.
- admaTableWords – ADMA table length.
- dataBufferAddr – Data buffer address.
- dataBytes – Data Data length.
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum *_usdhc_adma_flag*.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

status_t USDHC_SetADMA1Descriptor(uint32_t *admaTable, uint32_t admaTableWords, const uint32_t *dataBufferAddr, uint32_t dataBytes, uint32_t flags)

Sets the ADMA1 descriptor table configuration.

Parameters

- admaTable – ADMA table address.
- admaTableWords – ADMA table length.
- dataBufferAddr – Data buffer address.
- dataBytes – Data length.
- flags – ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum *_usdhc_adma_flag*.

Return values

- kStatus_OutOfRange – ADMA descriptor table length isn't enough to describe data.
- kStatus_Success – Operate successfully.

static inline void USDHC_EnableInternalDMA(USDHC_Type *base, bool enable)

Enables internal DMA.

Parameters

- base – USDHC peripheral base address.
- enable – enable or disable flag

static inline void USDHC_EnableInterruptStatus(USDHC_Type *base, uint32_t mask)
Enables the interrupt status.

Parameters

- base – USDHC peripheral base address.
- mask – Interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline void USDHC_DisableInterruptStatus(USDHC_Type *base, uint32_t mask)
Disables the interrupt status.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline void USDHC_EnableInterruptSignal(USDHC_Type *base, uint32_t mask)
Enables the interrupt signal corresponding to the interrupt status flag.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline void USDHC_DisableInterruptSignal(USDHC_Type *base, uint32_t mask)
Disables the interrupt signal corresponding to the interrupt status flag.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline uint32_t USDHC_GetEnabledInterruptStatusFlags(USDHC_Type *base)
Gets the enabled interrupt status.

Parameters

- base – USDHC peripheral base address.

Returns
Current interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline uint32_t USDHC_GetInterruptStatusFlags(USDHC_Type *base)
Gets the current interrupt status.

Parameters

- base – USDHC peripheral base address.

Returns
Current interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline void USDHC_ClearInterruptStatusFlags(USDHC_Type *base, uint32_t mask)
Clears a specified interrupt status. write 1 clears.

Parameters

- base – USDHC peripheral base address.
- mask – The interrupt status flags mask(*_usdhc_interrupt_status_flag*).

static inline uint32_t USDHC_GetAutoCommand12ErrorStatusFlags(USDHC_Type *base)
Gets the status of auto command 12 error.

Parameters

- base – USDHC peripheral base address.

Returns
Auto command 12 error status flags mask(*_usdhc_auto_command12_error_status_flag*).

static inline uint32_t USDHC_GetAdmaErrorStatusFlags(USDHC_Type *base)

Gets the status of the ADMA error.

Parameters

- base – USDHC peripheral base address.

Returns
ADMA error status flags mask(*_usdhc_adma_error_status_flag*).

static inline uint32_t USDHC_GetPresentStatusFlags(USDHC_Type *base)

Gets a present status.

This function gets the present USDHC's status except for an interrupt status and an error status.

Parameters

- base – USDHC peripheral base address.

Returns
Present USDHC's status flags mask(*_usdhc_present_status_flag*).

void USDHC_GetCapability(USDHC_Type *base, *usdhc_capability_t* *capability)

Gets the capability information.

Parameters

- base – USDHC peripheral base address.
- capability – Structure to save capability information.

static inline void USDHC_ForceClockOn(USDHC_Type *base, bool enable)

Forces the card clock on.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

uint32_t USDHC_SetSdClock(USDHC_Type *base, uint32_t srcClock_Hz, uint32_t busClock_Hz)

Sets the SD bus clock frequency.

Parameters

- base – USDHC peripheral base address.
- srcClock_Hz – USDHC source clock frequency united in Hz.
- busClock_Hz – SD bus clock frequency united in Hz.

Returns
The nearest frequency of busClock_Hz configured for SD bus.

bool USDHC_SetCardActive(USDHC_Type *base, uint32_t timeout)

Sends 80 clocks to the card to set it to the active state.

This function must be called each time the card is inserted to ensure that the card can receive the command correctly.

Parameters

- base – USDHC peripheral base address.
- timeout – Timeout to initialize card.

Return values

- true – Set card active successfully.
- false – Set card active failed.

static inline void USDHC_AssertHardwareReset(USDHC_Type *base, bool high)

Triggers a hardware reset.

Parameters

- base – USDHC peripheral base address.
- high – 1 or 0 level

static inline void USDHC_SetDataBusWidth(USDHC_Type *base, *usdhc_data_bus_width_t* width)

Sets the data transfer width.

Parameters

- base – USDHC peripheral base address.
- width – Data transfer width.

static inline void USDHC_WriteData(USDHC_Type *base, uint32_t data)

Fills the data port.

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

- base – USDHC peripheral base address.
- data – The data about to be sent.

static inline uint32_t USDHC_ReadData(USDHC_Type *base)

Retrieves the data from the data port.

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

- base – USDHC peripheral base address.

Returns

The data has been read.

void USDHC_SendCommand(USDHC_Type *base, *usdhc_command_t* *command)

Sends command function.

Parameters

- base – USDHC peripheral base address.
- command – configuration

static inline void USDHC_EnableWakeupEvent(USDHC_Type *base, uint32_t mask, bool enable)

Enables or disables a wakeup event in low-power mode.

Parameters

- base – USDHC peripheral base address.
- mask – Wakeup events mask(*_usdhc_wakeup_event*).
- enable – True to enable, false to disable.

static inline void USDHC_CardDetectByData3(USDHC_Type *base, bool enable)

Detects card insert status.

Parameters

- base – USDHC peripheral base address.

- enable – enable/disable flag

static inline bool USDHC_DetectCardInsert(USDHC_Type *base)

Detects card insert status.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_EnableSdioControl(USDHC_Type *base, uint32_t mask, bool enable)

Enables or disables the SDIO card control.

Parameters

- base – USDHC peripheral base address.
- mask – SDIO card control flags mask(*_usdhc_sdio_control_flag*).
- enable – True to enable, false to disable.

static inline void USDHC_SetContinueRequest(USDHC_Type *base)

Restarts a transaction which has stopped at the block GAP for the SDIO card.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_RequestStopAtBlockGap(USDHC_Type *base, bool enable)

Request stop at block gap function.

Parameters

- base – USDHC peripheral base address.
- enable – True to stop at block gap, false to normal transfer.

void USDHC_SetMmcBootConfig(USDHC_Type *base, const *usdhc_boot_config_t* *config)

Configures the MMC boot feature.

Example:

```
usdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kUSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
USDHC_SetMmcBootConfig(USDHC, &config);
```

Parameters

- base – USDHC peripheral base address.
- config – The MMC boot configuration information.

static inline void USDHC_EnableMmcBoot(USDHC_Type *base, bool enable)

Enables or disables the mmc boot mode.

Parameters

- base – USDHC peripheral base address.
- enable – True to enable, false to disable.

static inline void USDHC_SetForceEvent(USDHC_Type *base, uint32_t mask)

Forces generating events according to the given mask.

Parameters

- base – USDHC peripheral base address.
- mask – The force events bit posistion (_usdhc_force_event).

static inline bool USDHC_RequestTuningForSDR50(USDHC_Type *base)

Checks the SDR50 mode request tuning bit. When this bit set, application shall perform tuning for SDR50 mode.

Parameters

- base – USDHC peripheral base address.

static inline bool USDHC_RequestReTuning(USDHC_Type *base)

Checks the request re-tuning bit. When this bit is set, user should do manual tuning or standard tuning function.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_EnableAutoTuning(USDHC_Type *base, bool enable)

The SDR104 mode auto tuning enable and disable. This function should be called after tuning function execute pass, auto tuning will handle by hardware.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

void USDHC_EnableAutoTuningForCmdAndData(USDHC_Type *base)

The auto tuning enbale for CMD/DATA line.

Parameters

- base – USDHC peripheral base address.

void USDHC_EnableManualTuning(USDHC_Type *base, bool enable)

Manual tuning trigger or abort. User should handle the tuning cmd and find the boundary of the delay then calucate a average value which will be configured to the **CLK_TUNE_CTRL_STATUS** This function should be called before function *USDHC_AdjustDelayForManualTuning*.

Parameters

- base – USDHC peripheral base address.
- enable – tuning enable flag

static inline uint32_t USDHC_GetTuningDelayStatus(USDHC_Type *base)

Get the tuning delay cell setting.

Parameters

- base – USDHC peripheral base address.

Return values

CLK – Tuning Control and Status register value.

status_t USDHC_SetTuningDelay(USDHC_Type *base, uint32_t preDelay, uint32_t outDelay, uint32_t postDelay)

The tuning delay cell setting.

Parameters

- base – USDHC peripheral base address.
- preDelay – Set the number of delay cells on the feedback clock between the feedback clock and CLK_PRE.
- outDelay – Set the number of delay cells on the feedback clock between CLK_PRE and CLK_OUT.
- postDelay – Set the number of delay cells on the feedback clock between CLK_OUT and CLK_POST.

Return values

- kStatus_Fail – config the delay setting fail
- kStatus_Success – config the delay setting success

status_t USDHC_AdjustDelayForManualTuning(USDHC_Type *base, uint32_t delay)

Adjusts delay for mannual tuning.

Deprecated:

Do not use this function. It has been superceded by USDHC_SetTuingDelay

Parameters

- base – USDHC peripheral base address.
- delay – setting configuration

Return values

- kStatus_Fail – config the delay setting fail
- kStatus_Success – config the delay setting success

static inline void USDHC_SetStandardTuningCounter(USDHC_Type *base, uint8_t counter)

set tuning counter tuning.

Parameters

- base – USDHC peripheral base address.
- counter – tuning counter

Return values

- kStatus_Fail – config the delay setting fail
- kStatus_Success – config the delay setting success

void USDHC_EnableStandardTuning(USDHC_Type *base, uint32_t tuningStartTap, uint32_t step, bool enable)

The enable standard tuning function. The standard tuning window and tuning counter using the default config tuning cmd is sent by the software, user need to check whether the tuning result can be used for SDR50, SDR104, and HS200 mode tuning.

Parameters

- base – USDHC peripheral base address.
- tuningStartTap – start tap
- step – tuning step
- enable – enable/disable flag

static inline uint32_t USDHC_GetExecuteStdTuningStatus(USDHC_Type *base)

Gets execute STD tuning status.

Parameters

- base – USDHC peripheral base address.

static inline uint32_t USDHC_CheckStdTuningResult(USDHC_Type *base)

Checks STD tuning result.

Parameters

- base – USDHC peripheral base address.

static inline uint32_t USDHC_CheckTuningError(USDHC_Type *base)

Checks tuning error.

Parameters

- base – USDHC peripheral base address.

void USDHC_EnableDDRMode(USDHC_Type *base, bool enable, uint32_t nibblePos)

The enable/disable DDR mode.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag
- nibblePos – nibble position

static inline void USDHC_EnableHS400Mode(USDHC_Type *base, bool enable)

The enable/disable HS400 mode.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

static inline void USDHC_ResetStrobeDLL(USDHC_Type *base)

Resets the strobe DLL.

Parameters

- base – USDHC peripheral base address.

static inline void USDHC_EnableStrobeDLL(USDHC_Type *base, bool enable)

Enables/disables the strobe DLL.

Parameters

- base – USDHC peripheral base address.
- enable – enable/disable flag

void USDHC_ConfigStrobeDLL(USDHC_Type *base, uint32_t delayTarget, uint32_t updateInterval)

Configs the strobe DLL delay target and update interval.

Parameters

- base – USDHC peripheral base address.
- delayTarget – delay target
- updateInterval – update interval

static inline void USDHC_SetStrobeDllOverride(USDHC_Type *base, uint32_t delayTaps)

Enables manual override for slave delay chain using **STROBE_SLV_OVERRIDE_VAL**.

Parameters

- base – USDHC peripheral base address.
- delayTaps – Valid delay taps range from 1 - 128 taps. A value of 0 selects tap 1, and a value of 0x7F selects tap 128.

static inline uint32_t USDHC_GetStrobeDLLStatus(USDHC_Type *base)

Gets the strobe DLL status.

Parameters

- base – USDHC peripheral base address.

void USDHC_SetDataConfig(USDHC_Type *base, *usdhc_transfer_direction_t* dataDirection, uint32_t blockCount, uint32_t blockSize)

USDHC data configuration.

Parameters

- base – USDHC peripheral base address.
- dataDirection – Data direction, tx or rx.
- blockCount – Data block count.
- blockSize – Data block size.

void USDHC_TransferCreateHandle(USDHC_Type *base, *usdhc_handle_t* *handle, const *usdhc_transfer_callback_t* *callback, void *userData)

Creates the USDHC handle.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle pointer.
- callback – Structure pointer to contain all callback functions.
- userData – Callback function parameter.

status_t USDHC_TransferNonBlocking(USDHC_Type *base, *usdhc_handle_t* *handle, *usdhc_adma_config_t* *dmaConfig, *usdhc_transfer_t* *transfer)

Transfers the command/data using an interrupt and an asynchronous method.

This function sends a command and data and returns immediately. It doesn't wait for the transfer to complete or to encounter an error. The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note: Call API *USDHC_TransferCreateHandle* when calling this API.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle.
- dmaConfig – ADMA configuration.
- transfer – Transfer content.

Return values

- kStatus_InvalidArgument – Argument is invalid.
- kStatus_USDHC_BusyTransferring – Busy transferring.
- kStatus_USDHC_PrepareAdmaDescriptorFailed – Prepare ADMA descriptor failed.
- kStatus_Success – Operate successfully.

status_t USDHC_TransferBlocking(USDHC_Type *base, *usdhc_adma_config_t* *dmaConfig, *usdhc_transfer_t* *transfer)

Transfers the command/data using a blocking method.

This function waits until the command response/data is received or the USDHC encounters an error by polling the status flag.

The application must not call this API in multiple threads at the same time. Because this API doesn't support the re-entry mechanism.

Note: There is no need to call API *USDHC_TransferCreateHandle* when calling this API.

Parameters

- base – USDHC peripheral base address.
- dmaConfig – adma configuration
- transfer – Transfer content.

Return values

- kStatus_InvalidArgument – Argument is invalid.
- kStatus_USDHC_PrepareAdmaDescriptorFailed – Prepare ADMA descriptor failed.
- kStatus_USDHC_SendCommandFailed – Send command failed.
- kStatus_USDHC_TransferDataFailed – Transfer data failed.
- kStatus_Success – Operate successfully.

void USDHC_TransferHandleIRQ(USDHC_Type *base, *usdhc_handle_t* *handle)

IRQ handler for the USDHC.

This function deals with the IRQs on the given host controller.

Parameters

- base – USDHC peripheral base address.
- handle – USDHC handle.

void USDHC_DriverIRQHandler(uint32_t instance)

FSL_USDHC_DRIVER_VERSION

Driver version.

Enum _usdhc_status. USDHC status.

Values:

enumerator kStatus_USDHC_BusyTransferring
Transfer is on-going.

enumerator kStatus_USDHC_PrepareAdmaDescriptorFailed
Set DMA descriptor failed.

enumerator kStatus_USDHC_SendCommandFailed
Send command failed.

enumerator kStatus_USDHC_TransferDataFailed
Transfer data failed.

enumerator kStatus_USDHC_DMADataAddrNotAlign
Data address not aligned.

enumerator kStatus_USDHC_ReTuningRequest
Re-tuning request.

enumerator kStatus_USDHC_TuningError
Tuning error.

enumerator kStatus_USDHC_NotSupport
Not support.

enumerator kStatus_USDHC_TransferDataComplete
Transfer data complete.

enumerator kStatus_USDHC_SendCommandSuccess
Transfer command complete.

enumerator kStatus_USDHC_TransferDMAComplete
Transfer DMA complete.

Enum _usdhc_capability_flag. Host controller capabilities flag mask. .

Values:

enumerator kUSDHC_SupportAdmaFlag
Support ADMA.

enumerator kUSDHC_SupportHighSpeedFlag
Support high-speed.

enumerator kUSDHC_SupportDmaFlag
Support DMA.

enumerator kUSDHC_SupportSuspendResumeFlag
Support suspend/resume.

enumerator kUSDHC_SupportV330Flag
Support voltage 3.3V.

enumerator kUSDHC_SupportV300Flag
Support voltage 3.0V.

enumerator kUSDHC_Support4BitFlag
Flag in HTCAPBLT_MBL's position, supporting 4-bit mode.

enumerator kUSDHC_Support8BitFlag
Flag in HTCAPBLT_MBL's position, supporting 8-bit mode.

enumerator kUSDHC_SupportDDR50Flag
SD version 3.0 new feature, supporting DDR50 mode.

enumerator kUSDHC_SupportSDR104Flag
Support SDR104 mode.

enumerator kUSDHC_SupportSDR50Flag
Support SDR50 mode.

Enum _usdhc_wakeup_event. Wakeup event mask. .

Values:

enumerator kUSDHC_WakeupEventOnCardInt
Wakeup on card interrupt.

enumerator kUSDHC_WakeupEventOnCardInsert
Wakeup on card insertion.

enumerator kUSDHC_WakeupEventOnCardRemove
Wakeup on card removal.

enumerator kUSDHC_WakeupEventsAll
All wakeup events

Enum _usdhc_reset. Reset type mask. .

Values:

enumerator kUSDHC_ResetAll
Reset all except card detection.

enumerator kUSDHC_ResetCommand
Reset command line.

enumerator kUSDHC_ResetData
Reset data line.

enumerator kUSDHC_ResetTuning
Reset tuning circuit.

enumerator kUSDHC_ResetsAll
All reset types

Enum _usdhc_transfer_flag. Transfer flag mask.
Values:

enumerator kUSDHC_EnableDmaFlag
Enable DMA.

enumerator kUSDHC_CommandTypeSuspendFlag
Suspend command.

enumerator kUSDHC_CommandTypeResumeFlag
Resume command.

enumerator kUSDHC_CommandTypeAbortFlag
Abort command.

enumerator kUSDHC_EnableBlockCountFlag
Enable block count.

enumerator kUSDHC_EnableAutoCommand12Flag
Enable auto CMD12.

enumerator kUSDHC_DataReadFlag
Enable data read.

enumerator kUSDHC_MultipleBlockFlag
Multiple block data read/write.

enumerator kUSDHC_EnableAutoCommand23Flag
Enable auto CMD23.

enumerator kUSDHC_ResponseLength136Flag
136-bit response length.

enumerator kUSDHC_ResponseLength48Flag
48-bit response length.

enumerator kUSDHC_ResponseLength48BusyFlag
48-bit response length with busy status.

enumerator kUSDHC_EnableCrcCheckFlag
Enable CRC check.

enumerator kUSDHC_EnableIndexCheckFlag
Enable index check.

enumerator kUSDHC_DataPresentFlag
Data present flag.

Enum _usdhc_present_status_flag. Present status flag mask. .

Values:

enumerator kUSDHC_CommandInhibitFlag
Command inhibit.

enumerator kUSDHC_DataInhibitFlag
Data inhibit.

enumerator kUSDHC_DataLineActiveFlag
Data line active.

enumerator kUSDHC_SdClockStableFlag
SD bus clock stable.

enumerator kUSDHC_WriteTransferActiveFlag
Write transfer active.

enumerator kUSDHC_ReadTransferActiveFlag
Read transfer active.

enumerator kUSDHC_BufferWriteEnableFlag
Buffer write enable.

enumerator kUSDHC_BufferReadEnableFlag
Buffer read enable.

enumerator kUSDHC_ReTuningRequestFlag
Re-tuning request flag, only used for SDR104 mode.

enumerator kUSDHC_DelaySettingFinishedFlag
Delay setting finished flag.

enumerator kUSDHC_CardInsertedFlag
Card inserted.

enumerator kUSDHC_CommandLineLevelFlag
Command line signal level.

enumerator kUSDHC_Data0LineLevelFlag
Data0 line signal level.

enumerator kUSDHC_Data1LineLevelFlag
Data1 line signal level.

enumerator kUSDHC_Data2LineLevelFlag
Data2 line signal level.

enumerator kUSDHC_Data3LineLevelFlag
Data3 line signal level.

enumerator kUSDHC_Data4LineLevelFlag
Data4 line signal level.

enumerator kUSDHC_Data5LineLevelFlag
Data5 line signal level.

enumerator kUSDHC_Data6LineLevelFlag
Data6 line signal level.

enumerator kUSDHC_Data7LineLevelFlag
Data7 line signal level.

Enum _usdhc_interrupt_status_flag. Interrupt status flag mask. .

Values:

enumerator kUSDHC_CommandCompleteFlag
Command complete.

enumerator kUSDHC_DataCompleteFlag
Data complete.

enumerator kUSDHC_BlockGapEventFlag
Block gap event.

enumerator kUSDHC_DmaCompleteFlag
DMA interrupt.

enumerator kUSDHC_BufferWriteReadyFlag
Buffer write ready.

enumerator kUSDHC_BufferReadReadyFlag
Buffer read ready.

enumerator kUSDHC_CardInsertionFlag
Card inserted.

enumerator kUSDHC_CardRemovalFlag
Card removed.

enumerator kUSDHC_CardInterruptFlag
Card interrupt.

enumerator kUSDHC_ReTuningEventFlag
Re-Tuning event, only for SD3.0 SDR104 mode.

enumerator kUSDHC_TuningPassFlag
SDR104 mode tuning pass flag.

enumerator kUSDHC_TuningErrorFlag
SDR104 tuning error flag.

enumerator kUSDHC_CommandTimeoutFlag
Command timeout error.

enumerator kUSDHC_CommandCrcErrorFlag
Command CRC error.

enumerator kUSDHC_CommandEndBitErrorFlag
Command end bit error.

enumerator kUSDHC_CommandIndexErrorFlag
Command index error.

enumerator kUSDHC_DataTimeoutFlag
Data timeout error.

enumerator kUSDHC_DataCrcErrorFlag
Data CRC error.

enumerator kUSDHC_DataEndBitErrorFlag
Data end bit error.

enumerator kUSDHC_AutoCommand12ErrorFlag
Auto CMD12 error.

enumerator kUSDHC_DmaErrorFlag
DMA error.

enumerator kUSDHC_CommandErrorFlag
Command error

enumerator kUSDHC_DataErrorFlag
Data error

enumerator kUSDHC_ErrorFlag
All error

enumerator kUSDHC_DataFlag
Data interrupts

enumerator kUSDHC_DataDMAFlag
Data interrupts

enumerator kUSDHC_CommandFlag
Command interrupts

enumerator kUSDHC_CardDetectFlag
Card detection interrupts

enumerator kUSDHC_SDR104TuningFlag
SDR104 tuning flag.

enumerator kUSDHC_AllInterruptFlags
All flags mask

Enum _usdhc_auto_command12_error_status_flag. Auto CMD12 error status flag mask. .
Values:

enumerator kUSDHC_AutoCommand12NotExecutedFlag
Not executed error.

enumerator kUSDHC_AutoCommand12TimeoutFlag
Timeout error.

enumerator kUSDHC_AutoCommand12EndBitErrorFlag
End bit error.

enumerator kUSDHC_AutoCommand12CrcErrorFlag
CRC error.

enumerator kUSDHC_AutoCommand12IndexErrorFlag
Index error.

enumerator kUSDHC_AutoCommand12NotIssuedFlag
Not issued error.

Enum _usdhc_standard_tuning. Standard tuning flag.
Values:

enumerator kUSDHC_ExecuteTuning
Used to start tuning procedure.

enumerator kUSDHC_TuningSampleClockSel
When **std_tuning_en** bit is set, this bit is used to select sampleing clock.

Enum _usdhc_adma_error_status_flag. ADMA error status flag mask. .

Values:

enumerator kUSDHC_AdmaLenghMismatchFlag
Length mismatch error.

enumerator kUSDHC_AdmaDescriptorErrorFlag
Descriptor error.

Enum _usdhc_adma_error_state. ADMA error state.

This state is the detail state when ADMA error has occurred.

Values:

enumerator kUSDHC_AdmaErrorStateStopDma
Stop DMA, previous location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateFetchDescriptor
Fetch descriptor, current location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateChangeAddress
Change address, no DMA error has occurred.

enumerator kUSDHC_AdmaErrorStateTransferData
Transfer data, previous location set in the ADMA system address is errored address.

enumerator kUSDHC_AdmaErrorStateInvalidLength
Invalid length in ADMA descriptor.

enumerator kUSDHC_AdmaErrorStateInvalidDescriptor
Invalid descriptor fetched by ADMA.

enumerator kUSDHC_AdmaErrorState
ADMA error state

Enum _usdhc_force_event. Force event bit position. .

Values:

enumerator kUSDHC_ForceEventAutoCommand12NotExecuted
Auto CMD12 not executed error.

enumerator kUSDHC_ForceEventAutoCommand12Timeout
Auto CMD12 timeout error.

enumerator kUSDHC_ForceEventAutoCommand12CrcError
Auto CMD12 CRC error.

enumerator kUSDHC_ForceEventEndBitError
Auto CMD12 end bit error.

enumerator kUSDHC_ForceEventAutoCommand12IndexError
Auto CMD12 index error.

enumerator kUSDHC_ForceEventAutoCommand12NotIssued
Auto CMD12 not issued error.

enumerator kUSDHC_ForceEventCommandTimeout
Command timeout error.

enumerator kUSDHC_ForceEventCommandCrcError
Command CRC error.

enumerator kUSDHC_ForceEventCommandEndBitError
Command end bit error.

enumerator kUSDHC_ForceEventCommandIndexError
Command index error.

enumerator kUSDHC_ForceEventDataTimeout
Data timeout error.

enumerator kUSDHC_ForceEventDataCrcError
Data CRC error.

enumerator kUSDHC_ForceEventDataEndBitError
Data end bit error.

enumerator kUSDHC_ForceEventAutoCommand12Error
Auto CMD12 error.

enumerator kUSDHC_ForceEventCardInt
Card interrupt.

enumerator kUSDHC_ForceEventDmaError
Dma error.

enumerator kUSDHC_ForceEventTuningError
Tuning error.

enumerator kUSDHC_ForceEventsAll
All force event flags mask.

enum _usdhc_transfer_direction
Data transfer direction.
Values:

enumerator kUSDHC_TransferDirectionReceive
USDHC transfer direction receive.

enumerator kUSDHC_TransferDirectionSend
USDHC transfer direction send.

enum _usdhc_data_bus_width
Data transfer width.
Values:

enumerator kUSDHC_DataBusWidth1Bit
1-bit mode

enumerator kUSDHC_DataBusWidth4Bit
4-bit mode

enumerator kUSDHC_DataBusWidth8Bit
8-bit mode

enum _usdhc_endian_mode
Endian mode.
Values:

enumerator kUSDHC_EndianModeBig
Big endian mode.

enumerator kUSDHC_EndianModeHalfWordBig
Half word big endian mode.

enumerator kUSDHC_EndianModeLittle
Little endian mode.

enum _usdhc_dma_mode
DMA mode.
Values:

enumerator kUSDHC_DmaModeSimple
External DMA.

enumerator kUSDHC_DmaModeAdma1
ADMA1 is selected.

enumerator kUSDHC_DmaModeAdma2
ADMA2 is selected.

enumerator kUSDHC_ExternalDMA
External DMA mode selected.

Enum _usdhc_sdio_control_flag. SDIO control flag mask. .

Values:

enumerator kUSDHC_StopAtBlockGapFlag
Stop at block gap.

enumerator kUSDHC_ReadWaitControlFlag
Read wait control.

enumerator kUSDHC_InterruptAtBlockGapFlag
Interrupt at block gap.

enumerator kUSDHC_ReadDoneNo8CLK
Read done without 8 clk for block gap.

enumerator kUSDHC_ExactBlockNumberReadFlag
Exact block number read.

enum _usdhc_boot_mode
MMC card boot mode.

Values:

enumerator kUSDHC_BootModeNormal
Normal boot

enumerator kUSDHC_BootModeAlternative
Alternative boot

enum _usdhc_card_command_type
The command type.

Values:

enumerator kCARD_CommandTypeNormal
Normal command

enumerator kCARD_CommandTypeSuspend
Suspend command

enumerator kCARD_CommandTypeResume
Resume command

enumerator kCARD_CommandTypeAbort
Abort command

enumerator kCARD_CommandTypeEmpty
Empty command

enum _usdhc_card_response_type
The command response type.

Defines the command response type from card to host controller.

Values:

enumerator kCARD_ResponseTypeNone
Response type: none

enumerator kCARD_ResponseTypeR1
Response type: R1

enumerator kCARD_ResponseTypeR1b
Response type: R1b

enumerator kCARD_ResponseTypeR2
Response type: R2

enumerator kCARD_ResponseTypeR3
Response type: R3

enumerator kCARD_ResponseTypeR4
Response type: R4

enumerator kCARD_ResponseTypeR5
Response type: R5

enumerator kCARD_ResponseTypeR5b
Response type: R5b

enumerator kCARD_ResponseTypeR6
Response type: R6

enumerator kCARD_ResponseTypeR7
Response type: R7

Enum _usdhc_adma1_descriptor_flag. The mask for the control/status field in ADMA1 descriptor.

Values:

enumerator kUSDHC_Adma1DescriptorValidFlag
Valid flag.

enumerator kUSDHC_Adma1DescriptorEndFlag
End flag.

enumerator kUSDHC_Adma1DescriptorInterrupFlag
Interrupt flag.

enumerator kUSDHC_Adma1DescriptorActivity1Flag
Activity 1 flag.

enumerator kUSDHC_Adma1DescriptorActivity2Flag
Activity 2 flag.

enumerator kUSDHC_Adma1DescriptorTypeNop
No operation.

enumerator kUSDHC_Adma1DescriptorTypeTransfer
Transfer data.

enumerator kUSDHC_Adma1DescriptorTypeLink
Link descriptor.

enumerator kUSDHC_Adma1DescriptorTypeSetLength
Set data length.

Enum _usdhc_adma2_descriptor_flag. ADMA1 descriptor control and status mask.

Values:

enumerator kUSDHC_Adma2DescriptorValidFlag
Valid flag.

enumerator kUSDHC_Adma2DescriptorEndFlag
End flag.

enumerator kUSDHC_Adma2DescriptorInterruptFlag
Interrupt flag.

enumerator kUSDHC_Adma2DescriptorActivity1Flag
Activity 1 mask.

enumerator kUSDHC_Adma2DescriptorActivity2Flag
Activity 2 mask.

enumerator kUSDHC_Adma2DescriptorTypeNop
No operation.

enumerator kUSDHC_Adma2DescriptorTypeReserved
Reserved.

enumerator kUSDHC_Adma2DescriptorTypeTransfer

Transfer type.

enumerator kUSDHC_Adma2DescriptorTypeLink

Link type.

Enum _usdhc_adma_flag. ADMA descriptor configuration flag. .

Values:

enumerator kUSDHC_AdmaDescriptorSingleFlag

Try to finish the transfer in a single ADMA descriptor. If transfer size is bigger than one ADMA descriptor's ability, new another descriptor for data transfer.

enumerator kUSDHC_AdmaDescriptorMultipleFlag

Create multiple ADMA descriptors within the ADMA table, this is used for mmc boot mode specifically, which need to modify the ADMA descriptor on the fly, so the flag should be used combining with stop at block gap feature.

enum _usdhc_burst_len

DMA transfer burst len config.

Values:

enumerator kUSDHC_EnBurstLenForINCR

Enable burst len for INCR.

enumerator kUSDHC_EnBurstLenForINCR4816

Enable burst len for INCR4/INCR8/INCR16.

enumerator kUSDHC_EnBurstLenForINCR4816WRAP

Enable burst len for INCR4/8/16 WRAP.

Enum _usdhc_transfer_data_type. Tansfer data type definition.

Values:

enumerator kUSDHC_TransferDataNormal

Transfer normal read/write data.

enumerator kUSDHC_TransferDataTuning

Transfer tuning data.

enumerator kUSDHC_TransferDataBoot

Transfer boot data.

enumerator kUSDHC_TransferDataBootcontinous

Transfer boot data continuously.

typedef enum *_usdhc_transfer_direction* usdhc_transfer_direction_t
Data transfer direction.

typedef enum *_usdhc_data_bus_width* usdhc_data_bus_width_t
Data transfer width.

typedef enum *_usdhc_endian_mode* usdhc_endian_mode_t
Endian mode.

typedef enum *_usdhc_dma_mode* usdhc_dma_mode_t
DMA mode.

typedef enum *_usdhc_boot_mode* usdhc_boot_mode_t
MMC card boot mode.

typedef enum *_usdhc_card_command_type* usdhc_card_command_type_t
The command type.

typedef enum *_usdhc_card_response_type* usdhc_card_response_type_t
The command response type.
Defines the command response type from card to host controller.

typedef enum *_usdhc_burst_len* usdhc_burst_len_t
DMA transfer burst len config.

typedef uint32_t usdhc_adma1_descriptor_t
Defines the ADMA1 descriptor structure.

typedef struct *_usdhc_adma2_descriptor* usdhc_adma2_descriptor_t
Defines the ADMA2 descriptor structure.

typedef struct *_usdhc_capability* usdhc_capability_t
USDHC capability information.
Defines a structure to save the capability information of USDHC.

typedef struct *_usdhc_boot_config* usdhc_boot_config_t
Data structure to configure the MMC boot feature.

typedef struct *_usdhc_config* usdhc_config_t
Data structure to initialize the USDHC.

typedef struct *_usdhc_command* usdhc_command_t
Card command descriptor.
Defines card command-related attribute.

typedef struct *_usdhc_adma_config* usdhc_adma_config_t
ADMA configuration.

typedef struct *_usdhc_scatter_gather_data_list* usdhc_scatter_gather_data_list_t

Card scatter gather data list.

Allow application register uncontinuous data buffer for data transfer.

typedef struct *_usdhc_scatter_gather_data* usdhc_scatter_gather_data_t

Card scatter gather data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

typedef struct *_usdhc_scatter_gather_transfer* usdhc_scatter_gather_transfer_t

usdhc scatter gather transfer.

typedef struct *_usdhc_data* usdhc_data_t

Card data descriptor.

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

typedef struct *_usdhc_transfer* usdhc_transfer_t

Transfer state.

typedef struct *_usdhc_handle* usdhc_handle_t

USDHC handle typedef.

typedef struct *_usdhc_transfer_callback* usdhc_transfer_callback_t

USDHC callback functions.

typedef *status_t* (*usdhc_transfer_function_t)(USDHC_Type *base, *usdhc_transfer_t* *content)

USDHC transfer function.

typedef struct *_usdhc_host* usdhc_host_t

USDHC host descriptor.

USDHC_MAX_BLOCK_COUNT

Maximum block count can be set one time.

FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER

USDHC scatter gather feature control macro.

USDHC_ADMA1_ADDRESS_ALIGN

The alignment size for ADDRESS filed in ADMA1's descriptor.

USDHC_ADMA1_LENGTH_ALIGN

The alignment size for LENGTH field in ADMA1's descriptor.

USDHC_ADMA2_ADDRESS_ALIGN

The alignment size for ADDRESS field in ADMA2's descriptor.

USDHC_ADMA2_LENGTH_ALIGN

The alignment size for LENGTH filed in ADMA2's descriptor.

USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT

The bit shift for ADDRESS filed in ADMA1's descriptor.

Address/page field	Reserved	Attribute					
31 12	11 6	05	04	03	02	01	00
address or data length	000000	Act2	Act1	0	Int	End	Valid

Act2	Act1	Comment	31-28	27-12
0	0	No op	Don't care	
0	1	Set data length	0000	Data Length
1	0	Transfer data	Data address	
1	1	Link descriptor	Descriptor address	

USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK

The bit mask for ADDRESS field in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT

The bit shift for LENGTH filed in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK

The mask for LENGTH field in ADMA1's descriptor.

USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY

The maximum value of LENGTH filed in ADMA1's descriptor. Since the max transfer size ADMA1 support is 65535 which is indivisible by 4096, so to make sure a large data load transfer (>64KB) continuously (require the data address be always align with 4096), software will set the maximum data length for ADMA1 to (64 - 4)KB.

USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT

The bit shift for LENGTH field in ADMA2's descriptor.

Address field	Length	Reserved	Attribute					
63 32	31 16	15 06	05	04	03	02	01	00
32-bit address	16-bit length	0000000000	Act2	Act1	0	Int	End	Valid

Act2	Act1	Comment	Operation
0	0	No op	Don't care
0	1	Reserved	Read this line and go to next one
1	0	Transfer data	Transfer data with address and length set in this descriptor line
1	1	Link descriptor	Link to another descriptor

USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK

The bit mask for LENGTH field in ADMA2's descriptor.

USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY

The maximum value of LENGTH field in ADMA2's descriptor.

struct _usdhc_adma2_descriptor

#include <fsl_usdhc.h> Defines the ADMA2 descriptor structure.

Public Members

uint32_t attribute

The control and status field.

uint32_t address

The address field.

struct _usdhc_capability

#include <fsl_usdhc.h> USDHC capability information.

Defines a structure to save the capability information of USDHC.

Public Members

uint32_t sdVersion

Support SD card/sdio version.

uint32_t mmcVersion

Support EMMC card version.

uint32_t maxBlockLength

Maximum block length united as byte.

uint32_t maxBlockCount

Maximum block count can be set one time.

uint32_t flags

Capability flags to indicate the support information(*_usdhc_capability_flag*).

struct _usdhc_boot_config

#include <fsl_usdhc.h> Data structure to configure the MMC boot feature.

Public Members

uint32_t ackTimeoutCount

Timeout value for the boot ACK. The available range is 0 ~ 15.

usdhc_boot_mode_t bootMode

Boot mode selection.

uint32_t blockCount

Stop at block gap value of automatic mode. Available range is 0 ~ 65535.

size_t blockSize

Block size.

bool enableBootAck

Enable or disable boot ACK.

bool enableAutoStopAtBlockGap

Enable or disable auto stop at block gap function in boot period.

struct _usdhc_config

#include <fsl_usdhc.h> Data structure to initialize the USDHC.

Public Members

uint32_t dataTimeout

Data timeout value.

usdhc_endian_mode_t endianMode

Endian mode.

uint8_t readWatermarkLevel

Watermark level for DMA read operation. Available range is 1 ~ 128.

uint8_t writeWatermarkLevel

Watermark level for DMA write operation. Available range is 1 ~ 128.

struct _usdhc_command

#include <fsl_usdhc.h> Card command descriptor.

Defines card command-related attribute.

Public Members

uint32_t index
: Command index.

uint32_t argument
: Command argument.

usdhc_card_command_type_t type
: Command type.

usdhc_card_response_type_t responseType
: Command response type.

uint32_t response[4U]
: Response for this command.

uint32_t responseErrorFlags
: Response error flag, which need to check the command reponse.

uint32_t flags
: Cmd flags.

struct __usdhc_adma_config
: *#include <fsl_usdhc.h>* ADMA configuration.

Public Members

usdhc_dma_mode_t dmaMode
: DMA mode.

uint32_t *admaTable
: ADMA table address, can't be null if transfer way is ADMA1/ADMA2.

uint32_t admaTableWords
: ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.

struct __usdhc_scatter_gather_data_list
: *#include <fsl_usdhc.h>* Card scatter gather data list.

  Allow application register uncontinuous data buffer for data transfer.

struct __usdhc_scatter_gather_data
: *#include <fsl_usdhc.h>* Card scatter gather data descriptor.

  Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Public Members

bool enableAutoCommand12
: Enable auto CMD12.

bool enableAutoCommand23
: Enable auto CMD23.

bool enableIgnoreError
: Enable to ignore error event to read/write all the data.

usdhc_transfer_direction_t dataDirection
: data direction

uint8_t dataType
: this is used to distinguish the normal/tuning/boot data.

size_t blockSize
: Block size.

usdhc_scatter_gather_data_list_t sgData
: scatter gather data

struct __usdhc_scatter_gather_transfer
: *#include <fsl_usdhc.h>* usdhc scatter gather transfer.

Public Members

usdhc_scatter_gather_data_t *data
: Data to transfer.

usdhc_command_t *command
: Command to send.

struct __usdhc_data
: *#include <fsl_usdhc.h>* Card data descriptor.

  Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Public Members

bool enableAutoCommand12
: Enable auto CMD12.

bool enableAutoCommand23
: Enable auto CMD23.

bool enableIgnoreError
: Enable to ignore error event to read/write all the data.

uint8_t dataType
: this is used to distinguish the normal/tuning/boot data.

size_t blockSize
: Block size.

uint32_t blockCount
: Block count.

uint32_t *rxData
: Buffer to save data read.

const uint32_t *txData
: Data buffer to write.

struct __usdhc_transfer
: *#include <fsl_usdhc.h>* Transfer state.

Public Members

usdhc_data_t *data
: Data to transfer.

usdhc_command_t *command
: Command to send.

struct __usdhc_transfer_callback
: *#include <fsl_usdhc.h>* USDHC callback functions.

Public Members

void (*CardInserted)(USDHC_Type *base, void *userData)
: Card inserted occurs when DAT3/CD pin is for card detect

void (*CardRemoved)(USDHC_Type *base, void *userData)
: Card removed occurs

void (*SdioInterrupt)(USDHC_Type *base, void *userData)
: SDIO card interrupt occurs

void (*BlockGap)(USDHC_Type *base, void *userData)

stopped at block gap event

void (*TransferComplete)(USDHC_Type *base, *usdhc_handle_t* *handle, *status_t* status, void *userData)

Transfer complete callback.

void (*ReTuning)(USDHC_Type *base, void *userData)

Handle the re-tuning.

struct _usdhc_handle

#include <fsl_usdhc.h> USDHC handle.

Defines the structure to save the USDHC state information and callback function.

Note: All the fields except interruptFlags and transferredWords must be allocated by the user.

Public Members

usdhc_data_t *volatile data

Transfer parameter. Data to transfer.

usdhc_command_t *volatile command

Transfer parameter. Command to send.

volatile uint32_t transferredWords

Transfer status. Words transferred by DATAPORT way.

usdhc_transfer_callback_t callback

Callback function.

void *userData

Parameter for transfer complete callback.

bool enDMA

Transfer parameter. Was DMA used?

struct _usdhc_host

#include <fsl_usdhc.h> USDHC host descriptor.

Public Members

USDHC_Type *base

USDHC peripheral base address.

uint32_t sourceClock_Hz
: USDHC source clock frequency united in Hz.

usdhc_config_t config
: USDHC configuration.

usdhc_capability_t capability
: USDHC capability information.

usdhc_transfer_function_t transfer
: USDHC transfer function.

2.89 UTICK: MictoTick Timer Driver

void UTICK_Init(UTICK_Type *base)
: Initializes an UTICK by turning its bus clock on.

void UTICK_Deinit(UTICK_Type *base)
: Deinitializes a UTICK instance.

  This function shuts down Utick bus clock

  Parameters

  - base – UTICK peripheral base address.

uint32_t UTICK_GetStatusFlags(UTICK_Type *base)
: Get Status Flags.

  This returns the status flag

  Parameters

  - base – UTICK peripheral base address.

  Returns
  status register value

void UTICK_ClearStatusFlags(UTICK_Type *base)
: Clear Status Interrupt Flags.

  This clears intr status flag

  Parameters

  - base – UTICK peripheral base address.

  Returns
  none

void UTICK_SetTick(UTICK_Type *base, *utick_mode_t* mode, uint32_t count, *utick_callback_t* cb)
: Starts UTICK.

  This function starts a repeat/onetime countdown with an optional callback

  Parameters

  - base – UTICK peripheral base address.
  - mode – UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)

- count – UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
- cb – UTICK callback (can be left as NULL if none, otherwise should be a void func(void))

Returns
none

void UTICK_HandleIRQ(UTICK_Type *base, *utick_callback_t* cb)

UTICK Interrupt Service Handler.

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in *UTICK_SetTick()*). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

- base – UTICK peripheral base address.
- cb – callback scheduled for this instance of UTICK

Returns
none

FSL_UTICK_DRIVER_VERSION

UTICK driver version 2.0.6.

enum _utick_mode

UTICK timer operational mode.

Values:

enumerator kUTICK_Onetime

Trigger once

enumerator kUTICK_Repeat

Trigger repeatedly

typedef enum *_utick_mode* utick_mode_t

UTICK timer operational mode.

typedef void (*utick_callback_t)(void)

UTICK callback function.

2.90 WWDT: Windowed Watchdog Timer Driver

void WWDT_GetDefaultConfig(*wwdt_config_t* *config)

Initializes WWDT configure structure.

This function initializes the WWDT configure structure to default value. The default value are:

```
config->enableWwdt = true;
config->enableWatchdogReset = false;
config->enableWatchdogProtect = false;
config->enableLockOscillator = false;
config->windowValue = 0xFFFFFFU;
```

(continues on next page)

(continued from previous page)

```
config->timeoutValue = 0xFFFFFFU;
config->warningValue = 0;
```

See also:

wwdt_config_t

Parameters

- config – Pointer to WWDT config structure.

void WWDT_Init(WWDT_Type *base, const *wwdt_config_t* *config)

Initializes the WWDT.

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

Example:

```
wwdt_config_t config;
WWDT_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
WWDT_Init(wwdt_base,&config);
```

Parameters

- base – WWDT peripheral base address
- config – The configuration of WWDT

void WWDT_Deinit(WWDT_Type *base)

Shuts down the WWDT.

This function shuts down the WWDT.

Parameters

- base – WWDT peripheral base address

static inline void WWDT_Enable(WWDT_Type *base)

Enables the WWDT module.

This function write value into WWDT_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

Parameters

- base – WWDT peripheral base address

static inline void WWDT_Disable(WWDT_Type *base)

Disables the WWDT module.

Deprecated:

Do not use this function. It will be deleted in next release version, for once the bit field of WDEN written with a 1, it can not be re-written with a 0.

This function write value into WWDT_MOD register to disable the WWDT.

Parameters

- base – WWDT peripheral base address

static inline uint32_t WWDT_GetStatusFlags(WWDT_Type *base)

Gets all WWDT status flags.

This function gets all status flags.

Example for getting Timeout Flag:

```
uint32_t status;
status = WWDT_GetStatusFlags(wwdt_base) & kWWDT_TimeoutFlag;
```

Parameters

- base – WWDT peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration *_wwdt_status_flags_t*

void WWDT_ClearStatusFlags(WWDT_Type *base, uint32_t mask)

Clear WWDT flag.

This function clears WWDT status flag.

Example for clearing warning flag:

```
WWDT_ClearStatusFlags(wwdt_base, kWWDT_WarningFlag);
```

Parameters

- base – WWDT peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration *_wwdt_status_flags_t*

static inline void WWDT_SetWarningValue(WWDT_Type *base, uint32_t warningValue)

Set the WWDT warning value.

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

Parameters

- base – WWDT peripheral base address
- warningValue – WWDT warning value.

static inline void WWDT_SetTimeoutValue(WWDT_Type *base, uint32_t timeoutCount)

Set the WWDT timeout value.

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below 0xFF will cause 0xFF to be loaded into the TC register. Thus the minimum time-out interval is TWDCLK*256*4. If enableWatchdogProtect flag is true in wwdt_config_t config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the WDTOF flag.

Parameters

- base – WWDT peripheral base address
- timeoutCount – WWDT timeout value, count of WWDT clock tick.

static inline void WWDT_SetWindowValue(WWDT_Type *base, uint32_t windowValue)

Sets the WWDT window value.

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set windowValue to 0xFFFFFF (maximum possible timer value) so windowing is not in effect.

Parameters

- base – WWDT peripheral base address
- windowValue – WWDT window value.

void WWDT_Refresh(WWDT_Type *base)

Refreshes the WWDT timer.

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – WWDT peripheral base address

FSL_WWDT_DRIVER_VERSION

Defines WWDT driver version.

WWDT_FIRST_WORD_OF_REFRESH

First word of refresh sequence

WWDT_SECOND_WORD_OF_REFRESH

Second word of refresh sequence

enum _wwdt_status_flags_t

WWDT status flags.

This structure contains the WWDT status flags for use in the WWDT functions.

Values:

enumerator kWWDT_TimeoutFlag

Time-out flag, set when the timer times out

enumerator kWWDT_WarningFlag

Warning interrupt flag, set when timer is below the value WDWARNINT

typedef struct *_wwdt_config* wwdt_config_t

Describes WWDT configuration structure.

struct _wwdt_config

#include <fsl_wwdt.h> Describes WWDT configuration structure.

Public Members

bool enableWwdt

Enables or disables WWDT

bool enableWatchdogReset

true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset

bool enableWatchdogProtect

true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time

uint32_t windowValue

Window value, set this to 0xFFFFFF if windowing is not in effect

uint32_t timeoutValue

Timeout value

uint32_t warningValue

Watchdog time counter value that will generate a warning interrupt. Set this to 0 for no warning

uint32_t clockFreq_Hz

Watchdog clock source frequency.

2.91 CACHE: CACHE Memory Controller

uint32_t XCACHE_GetInstanceByAddr(uint32_t address)

brief Returns an instance number given physical memory address.

param address The physical memory address.

Returns

XCACHE instance number starting from 0.

void XCACHE_EnableCache(XCACHE_Type *base)

Enables the cache.

Parameters

- base – XCACHE peripheral base address.

void XCACHE_DisableCache(XCACHE_Type *base)

Disables the cache.

Parameters

- base – XCACHE peripheral base address.

void XCACHE_InvalidateCache(XCACHE_Type *base)

Invalidates the cache.

Parameters

- base – XCACHE peripheral base address.

void XCACHE_InvalidateCacheByRange(uint32_t address, uint32_t size_byte)

Invalidates cache by range.

Note: Address and size should be aligned to "XCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to XCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be invalidated, should be larger than 0, better to align with cache line size.

void XCACHE_CleanCache(XCACHE_Type *base)

Cleans the cache.

Parameters

- base – XCACHE peripheral base address.

void XCACHE_CleanCacheByRange(uint32_t address, uint32_t size_byte)

Cleans cache by range.

Note: Address and size should be aligned to "XCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to XCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be cleaned, should be larger than 0, better to align with cache line size.

void XCACHE_CleanInvalidateCache(XCACHE_Type *base)

Cleans and invalidates the cache.

Parameters

- base – XCACHE peripheral base address.

void XCACHE_CleanInvalidateCacheByRange(uint32_t address, uint32_t size_byte)

Cleans and invalidate cache by range.

Note: Address and size should be aligned to "XCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to XCACHE_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address of cache.
- size_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0, better to align with cache line size.

static inline void ICACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates instruction cache by range.

Note: Address and size should be aligned to XCACHE_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_XCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated, should be larger than 0, better to align with cache line size.

static inline void DCACHE_InvalidateByRange(uint32_t address, uint32_t size_byte)

Invalidates data cache by range.

Note: Address and size should be aligned to XCACHE_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_XCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be invalidated, should be larger than 0, better to align with cache line size.

static inline void DCACHE_CleanByRange(uint32_t address, uint32_t size_byte)

Clean data cache by range.

Note: Address and size should be aligned to XCACHE_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_XCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be cleaned, should be larger than 0, better to align with cache line size.

static inline void DCACHE_CleanInvalidateByRange(uint32_t address, uint32_t size_byte)

Cleans and Invalidates data cache by range.

Note: Address and size should be aligned to XCACHE_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_XCACHE_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application

should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Parameters

- address – The physical address.
- size_byte – size of the memory to be Cleaned and Invalidated, should be larger than 0, better to align with cache line size.

FSL_CACHE_DRIVER_VERSION

cache driver version.

XCACHE_LINESIZE_BYTE

cache line size.

2.92 XSPI: Serial Peripheral Interface Driver

Status structure of XSPI.

Values:

enumerator kStatus_XSPI_Busy

XSPI is Busy

enumerator kStatus_XSPI_IpCommandUsageError

XSPI Ip Command Usage Error

enumerator kStatus_XSPI_IpCommandtriggerError

XSPI Ip Command Trigger Error

enumerator kStatus_XSPI_IllegalInstructionError

XSPI Illegal Instruction Error

enumerator kStatus_XSPI_SequenceExecutionTimeout

XSPI Sequence Execution Timeout

enumerator kStatus_XSPI_FradCheckError

XSPI Frad Check Error

enumerator kStatus_XSPI_TgQueueWritingError

XSPI Tg Queue Writing Error

enumerator kStatus_XSPI_IpsBusTransError

XSPI Ip Bus Transfer Error

enumerator kStatus_XSPI_AhbReadAccessAsserted

AHB read access is asserted.

enumerator kStatus_XSPI_AhbWriteAccessAsserted
AHB write access is asserted.

enumerator kStatus_XSPI_IPAccessAsserted
IP access is asserted.

enumerator kStatus_XSPI_AhbSubBufferFactorError
Wrong sub buffer factor input.

enumerator kStatus_XSPI_RegWriteLocked
Fail to write register due to write operation is locked.

enumerator kStatus_XSPI_WaterMarkIllegal
The watermark value to set is not in multiple of 4 bytes.

enumerator kStatus_XSPI_PageProgramWaitFlagAsserted
The page program wait flag already asserted.

enumerator kStatus_XSPI_IpAccessAddrSettingInvalid
The input address for Ip access is invalid.

enumerator kStatus_XSPI_RxBufferEntriesCountError
The count of entries of RX buffer is incorrect.

enumerator kStatus_XSPI_IpAccessIPCRInvalid
Access attributes to write IPCR are not correct.

enumerator kStatus_XSPI_IpAccessNotGranted
IP request not granted by arbiter.

enumerator kStatus_XSPI_IpReadFinished
Finish IP read all data stored in buffer.

enumerator kStatus_XSPI_IpWriteFinished
Finish IP write operation.

enumerator kStatus_XSPI_TxBufferUnderrun
XSPI attempted to pull data when the TX buffer is empty.

enumerator kStatus_XSPI_ArbiterGranted
XSPI arbiter granted.

enum _xspi_lut_instr
CMD definition of XSPI, use to form LUT instruction, xspi_lut_instr_t.
Values:

enumerator kXSPI_Command_STOP
Stop execution, de-assert CS.

enumerator kXSPI_Command_SDR
Transmit Command code to Flash, using SDR mode.

enumerator kXSPI_Command_RADDR_SDR
Transmit Row Address to Flash, using SDR mode.

enumerator kXSPI_Command_DUMMY_SDR
Leave data lines un-driven by xSPI controller, using SDR mode.

enumerator kXSPI_Command_MODE_SDR
Transmit 8-bit Mode bits to Flash, using SDR mode.

enumerator kXSPI_Command_MODE2_SDR
Transmit 2-bit Mode bits to Flash, using SDR mode.

enumerator kXSPI_Command_MODE4_SDR
Transmit 4-bit Mode bits to Flash, using SDR mode.

enumerator kXSPI_Command_READ_SDR
Receive Read Data from Flash, using SDR mode.

enumerator kXSPI_Command_WRITE_SDR
Transmit Programming Data to Flash, using SDR mode.

enumerator kXSPI_Command_JMP_ON_CS
Stop execution, de-assert CS and save operand[7:0] as the instruction start pointer for next sequence

enumerator kXSPI_Command_RADDR_DDR
Transmit Row Address to Flash, using DDR mode.

enumerator kXSPI_Command_MODE_DDR
Transmit 8-bit Mode bits to Flash, using DDR mode.

enumerator kXSPI_Command_MODE2_DDR
Transmit 2-bit Mode bits to Flash, using DDR mode.

enumerator kXSPI_Command_MODE4_DDR
Transmit 4-bit Mode bits to Flash, using DDR mode.

enumerator kXSPI_Command_READ_DDR
Receive Read Data from Flash, using DDR mode.

enumerator kXSPI_Command_WRITE_DDR
Transmit Programming Data to Flash, using DDR mode.

enumerator kXSPI_Command_LEARN_DDR
Receive Read Data or Preamble bit from Flash, DDR mode.

enumerator kXSPI_Command_DDR
Transmit Command code to Flash, using DDR mode.

enumerator kXSPI_Command_CADDR_SDR
Transmit Column Address to Flash, using SDR mode.

enumerator kXSPI_Command_CADDR_DDR
Transmit Column Address to Flash, using DDR mode.

enumerator kXSPI_Command_JUMP_TO_SEQ
Jump the LUT sequence pointed to by the operand.

enum _xspi_pad
pad definition of XSPI, use to form LUT instruction.

Values:

enumerator kXSPI_1PAD
Transmit command/address and transmit/receive data only through DATA0/DATA1.

enumerator kXSPI_2PAD
Transmit command/address and transmit/receive data only through DATA[1:0].

enumerator kXSPI_4PAD
Transmit command/address and transmit/receive data only through DATA[3:0].

enumerator kXSPI_8PAD
Transmit command/address and transmit/receive data only through DATA[7:0].

enum _xspi_flags
XSPI interrupt status flags.

Deprecated:
For Error flag please use *xspi_error_flag_t* as instead, for cmd execution and arbitration flag please use *xspi_cmd_execution_arbitration_flag_t* as instead.

Values:

enumerator kXSPI_DataLearningFailedFlag
Data learning failed.

enumerator kXSPI_DllAbortFlag
DLL Abort.

enumerator kXSPI_DllUnlockFlag
This field is set whenever DLL unlock event occurs, irrespective of flash memory access.

enumerator kXSPI_TxBufferFillFlag
If the field remains cleared, the TX buffer can be considered as full.

enumerator kXSPI_TxBufferUnderrunFlag

This field is set if the module tries to pull data when the TX buffer is empty

enumerator kXSPI_RxBufferOverflowFlag

This field is set when no more data can be pushed into the RX buffer from the serial flash memory device.

enumerator kXSPI_RxBufferDrainFlag

This field is set if SR[RXWE] is asserted.

enumerator kXSPI_AhbAbortErrorFlag

This flag can be set when AHB transaction error

enumerator kXSPI_AhbIllegalTransactionErrorFlag

enumerator kXSPI_AhbIllegalBurstSizeErrorFlag

The total burst size is more than prefetch size.

enumerator kXSPI_AhbBufferOverflowFlag

The size of the AHB access exceed the size of the AHB buffer.

enumerator kXSPI_AhbPerformanceMonitorOverflowFlag

AHB performance hit or miss counter overflow.

enumerator kXSPI_AhbReadAddressErrorFlag

The master send an AHB read address to that buffer is not within the address range of any sub-buffers or the address within multiple sub-buffers.

enumerator kXSPI_IllegalInstructionErrorFlag

This field is set when an illegal instruction is encountered by the controller in any of the sequences.

enumerator kXSPI_IpCmdtriggerErrorFlag

Write access to Rx buffer control reg when IP-triggered command is executing.

enumerator kXSPI_PageProgramWaitFlag

Indicates assertion of the page-program wait flag after writing to flash memory.

enumerator kXSPI_IpCommandExecutionDoneFlag

Indicates XSPI has completed a running IP command.

enumerator kXSPI_SequenceExecutionTimeoutFlag

Sequence execution timeout.

enumerator kXSPI_FradMatchErrorFlag

enumerator kXSPI_Frad0AccessErrorFlag

enumerator kXSPI_Frad1AccessErrorFlag

enumerator kXSPI_Frad2AccessErrorFlag

enumerator kXSPI_Frad3AccessErrorFlag

enumerator kXSPI_Frad4AccessErrorFlag

enumerator kXSPI_Frad5AccessErrorFlag

enumerator kXSPI_Frad6AccessErrorFlag

enumerator kXSPI_Frad7AccessErrorFlag

enumerator kXSPI_Frad8AccessErrorFlag

enumerator kXSPI_Frad9AccessErrorFlag

enumerator kXSPI_Frad10AccessErrorFlag

enumerator kXSPI_Frad11AccessErrorFlag

enumerator kXSPI_Frad12AccessErrorFlag

enumerator kXSPI_Frad13AccessErrorFlag

enumerator kXSPI_Frad14AccessErrorFlag

enumerator kXSPI_Frad15AccessErrorFlag

enumerator kXSPI_FradnAccErrorFlag
ORed value of all frad error flag,.

enumerator kXSPI_IpsErrorFlag

enumerator kXSPI_TgnSfarErrorFlag

enumerator kXSPI_TgnIpcrErrorFlag

enumerator kXSPI_LockRegErrorFlag

enumerator kXSPI_ArbLockTimeoutFlag

enumerator kXSPI_ArbWinEventFlag

enum _xspi_error_flag

The enumeration of error flags. .

Values:

enumerator kXSPI_ErrorNoFradMatch

The transaction address does not lie within the address range of any FRAD.

enumerator kXSPI_ErrorFrad0Access

The transaction address lies within the address range of the FRAD0 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad1Access

The transaction address lies within the address range of the FRAD1 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad2Access

The transaction address lies within the address range of the FRAD2 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad3Access

The transaction address lies within the address range of the FRAD3 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad4Access

The transaction address lies within the address range of the FRAD4 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad5Access

The transaction address lies within the address range of the FRAD5 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad6Access

The transaction address lies within the address range of the FRAD6 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad7Access

The transaction address lies within the address range of the FRAD7 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock by another manager.

enumerator kXSPI_ErrorFrad8Access

enumerator kXSPI_ErrorFrad9Access

enumerator kXSPI_ErrorFrad10Access

enumerator kXSPI_ErrorFrad11Access

enumerator kXSPI_ErrorFrad12Access

enumerator kXSPI_ErrorFrad13Access

enumerator kXSPI_ErrorFrad14Access

enumerator kXSPI_ErrorFrad15Access

enumerator kXSPI_ErrorIpBusTransfer
A common error has occurred and XSPI has generated an IPS bus transfer error.

enumerator kXSPI_ErrorTg0Sfar
An SFAR write trigger an error while written to the TG0.

enumerator kXSPI_ErrorTg0Ipcr
An IPCR write trigger an error while written to the TG0.

enumerator kXSPI_ErrorTimeout
A flash memory transaction has triggered a timeout error and has been terminated by SFP.

enumerator kXSPI_ArbitrationWin
The target group request was granted access.

enumerator kXSPI_ErrorArbitrationLock
Arbitration lock has timed out.

enumerator kXSPI_ErrorWriteLockedReg
An attempt to write a locked register.

enumerator kXSPI_ErrorAllFlags

enum _xspi_cmd_execution_arbitration_flag
The enumeration of cmd execution and arbitration. .

Values:

enumerator kXSPI_FlagIpCmdFinished
IP command transaction finished flag.

enumerator kXSPI_FlagAhbReadAddrError
Wrong AHB read address, XSPI return AHB error payload.

enumerator kXSPI_FlagAhbPerformanceMonitorOverflow
Buffer hit or miss counter overflow.

enumerator kXSPI_FlagIpCmdTriggerFail
Fail when set IP RX buffer watermark.

enumerator kXSPI_FlagPageProgramWait
Indicate assertion of the page-program wait flag after writing to flash memory.

enumerator kXSPI_FlagAhbBufferOverflow
The size of the AHB access exceeds the size of AHB buffer.

enumerator kXSPI_FlagAhbIllegalBurstSizeError
The total burst size of the AHB transacton is greater than the prefetch data size.

enumerator kXSPI_FlagAhbIllegalTransactionError
XSPI has not generated a response to the AHB bus due to an illegal transaction and expiration of the watchdog timer.

enumerator kXSPI_FlagAhbTransactionError
AHB transaction error.

enumerator kXSPI_FlagRxBufferWatermarkExceeded
The RX buffer watermark has been exceeded.

enumerator kXSPI_FlagRxBufferOverflow
The RX buffer cannot receive any more data from the SFM device.

enumerator kXSPI_FlagIllegalInstrError
The xspi encountered an illegal instruction in any sequence.

enumerator kXSPI_FlagDllUnlock
DLL unlock.

enumerator kXSPI_FlagTxBufferUnderRun
XSPI attempt to pull data when the TX buffer is empty.

enumerator kXSPI_FlagTxBufferFill
The TX buffer is filling.

enumerator kXSPI_FlagDllAbort
DLL terminate.

enumerator kXSPI_FlagDataLearningPatternFailure
The XSPI encountered a DATA_LEARN instruction in a sequence, but no sampling point is found for the data learning pattern.

enum _xspi_interrupt_enable

Values:

enumerator kXSPI_TransactionFinishIntEnable
Enable generation of an interrupt when an IP command transaction completes.

enumerator kXSPI_AhbReadAddrErrorIntEnable
Enable generation of an interrupt on occurrence of an AHB read address error.

enumerator kXSPI_AhbPerformanceMonitorOverflowIntEnable
Enable generation of an interrupt on a buffer hit or miss counter overflow.

enumerator kXSPI_IpCmdTrigFailErrorIntEnable
IP command trigger fail error interrupt enable.

enumerator kXSPI_PageProgramWaitIntEnable
Enable generation of an interrupt on page-program wait.

enumerator kXSPI_AhbBufferOverflowIntEnable
Enable generation of an interrupt on occurrence of an AHB buffer overflow.

enumerator kXSPI_AhbIllegalBurstSizeErrorIntEnable
Enable generation of an interrupt on occurrence of an AHB illegal burst size error.

enumerator kXSPI_AhbIllegalTransactionErrorIntEnable
Enable generation of an interrupt of an AHB illegal transaction error.

enumerator kXSPI_AhbAbortErrorIntEnable
Enable generation of an interrupt on occurrence of an AHB abort error.

enumerator kXSPI_RxBufferDrainIntEnable
Enable generation of an interrupt on occurrence of RX buffer drain.

enumerator kXSPI_RxBufferOverflowIntEnable
Enable generation of an interrupt on occurrence of RX buffer overflow.

enumerator kXSPI_IllegalInstrErrorIntEnable
Enable generation of an interrupt on occurrence of an illegal instruction error.

enumerator kXSPI_DllUnlockIntEnable
Enable generation of an interrupt on a DLL unlock event.

enumerator kXSPI_TxBufferUnderrunIntEnable
Enable generation of an interrupt on TX buffer underrun.

enumerator kXSPI_TxBufferFillIntEnable
Enable generation of an interrupt on TX buffer fill.

enumerator kXSPI_DataLearningPatternFailIntEnable
Enable generation of an interrupt on data learning pattern failure.

enumerator kXSPI_NoFradMatchErrorIntEnable
Enable the interrupt generated when the transaction address does not lie within the address range of any FRAD.

enumerator kXSPI_Frad0AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD0 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad1AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD1 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad2AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD2 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad3AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD3 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad4AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD4 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad5AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD5 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad6AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD6 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad7AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD7 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad8AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD0 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad9AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD9 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad10AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address

range of the FRAD10 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad11AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD11 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad12AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD12 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad13AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD13 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad14AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD14 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_Frad15AccessErrorIntEnable

Enable the interrupt generated when the transaction address lies within the address range of the FRAD15 but it does not pass the access permission checks for the FRAD or the FRAD is under exclusive lock.

enumerator kXSPI_IpsErrIntEnable

Enable the interrupt generated when a common error has occurred an XSPI has generated an IPS bus transfer error.

enumerator kXSPI_Tg0SfarErrIntEnable

Enable the interrupt generated when an SFAR write triggers an error while being written to the target group 0.

enumerator kXSPI_Tg0IpcrErrIntEnable

Enable the interrupt generated when an IPCR write triggers an error while being written to the target group 0.

enumerator kXSPI_TimeoutErrIntEnable

Enable the interrupt generated when a flash memory transaction triggers a timeout error and has been terminated by SFP.

enumerator kXSPI_ArbitrationWinIntEnable

Enable the interrupt generated when the target group queue request is granted access.

enumerator kXSPI_ArbitrationLockTimeoutErrIntEnable

Enable the interrupt generated when a flash memory transaction triggers a timeout error and has been terminated by SFP.

enumerator kXSPI_LockRegWriteErrIntEnable
Enable the interrupt generated by an attempt to write a locked register.

enum _xspi_external_signal
The enumeration of Idle signal drive IOFA[3] and IOFA[2].

Values:

enumerator XSPI_SignalIOFA2
Idle signal drive IOFA[2].

enumerator XSPI_SignalIOFA3
Idle signal drive IOFA[3].

enum _xspi_dll_mode
Dll Mode enumerations.

Values:

enumerator kXSPI_AutoUpdateMode
Dll mode : Auto update mode.

enumerator kXSPI_BypassMode
Dll mode : Bypass mode.

enum _xspi_sample_clk_source
The enumeration of sample clock source which is used to sample data from external devices.

Note: If external device support Data strobe signal, the sample clock should selected as kXSPI_SampleClkFromExternalDQS.

Values:

enumerator kXSPI_SampleClkFromNonInvertedFullySpeedDummyPadLoopback
Sample clock from non inverted fully speed dummy pad loopback.

enumerator kXSPI_SampleClkFromInvertedFullySpeedDummyPadLoopback
Sample clock from inverted fully speed dummy pad loopback.

enumerator kXSPI_SampleClkFromHalfSpeedDummyPadLoopback
Sample clock from half fully speed dummy pad loopback.

enumerator kXSPI_SampleClkFromDqsPadLoopback
Sample clock from Dqs pad loopback.

enumerator kXSPI_SampleClkFromExternalDQS
Sample clock from external DQS signal.

enum _xspi_data_learning_pad_select

The selection of data learning pad.

Values:

enumerator kXSPI_OnlyIO1

IO1 selected as pad to use for pattern matching IO.

enumerator kXSPI_BothIO1IO3

IO1 and IO3 selected as pad to use for pattern matching IO.

enum _xspi_device_addr_mode

External device's address mode.

Values:

enumerator kXSPI_DeviceByteAddressable

Byte addressable.

enumerator kXSPI_Device2ByteAddressable

Two bytes addressable.

enumerator kXSPI_Device4ByteAddressable

Four bytes addressable.

enum _xspi_device_interface_type

The enumeration of external device's interface type, hyper bus or SPI(including strandard and extended).

Note: Different interface result different settings.

Values:

enumerator kXSPI_StrandardExtendedSPI

External device follow SPI protocol, DQS_OUT_EN must be 0

enumerator kXSPI_HyperBus

External device follow hyper Bus protocol, DQS_OUT_EN must be 1

enum _xspi_hyper_bus_x16_mode

The enumeration of hyper bus X16 mode.

Values:

enumerator kXSPI_x16ModeDisable

X16 mode is disabled.

enumerator kXSPI_x16ModeEnabledOnlyData

X16 mode is enabled only for data.

enumerator kXSPI_x16ModeEnabledAddrCmdData

X16 mode is enabled for address, command, and data.

enum _xspi_ddr_data_aligned_clk

Enumeration defining the alignment modes of XSPI DDR data with respect to the internal reference clock.

Values:

enumerator kXSPI_DDRDataAlignedWithInternalRefClk

DDR data aligned with internal reference clock.

enumerator kXSPI_DDRDataAlignedWith2xInternalRefClk

DDR data aligned with 2X internal reference clock.

enum _xspi_command_type

Command type.

Values:

enumerator kXSPI_Command

XSPI operation: Only command, both TX and Rx buffer are ignored.

enumerator kXSPI_Config

XSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

enumerator kXSPI_Read

XSPI operation: Read, only Rx Buffer is effective.

enumerator kXSPI_Write

XSPI operation: Read, only Tx Buffer is effective.

enum _xspi_mdad_mask_type

The enumeration of SFP MDAD mask type.

Values:

enumerator kXSPI_MdadMaskTypeAnd

Target master is mask & reference.

enumerator kXSPI_MdadMaskTypeOr

Target master is mask | reference.

enum _xspi_mdad_error_reason

The enumeration of MDAD error reasons.

Values:

enumerator kXSPI_ErrorOnlyDataSize

Error reason: Only Data size without required attributes.

enumerator kXSPI_ErrorOnlySeqId

Error reason: Only sequence ID without required attributes.

enumerator kXSPI_ErrorBothDataSizeSeqId

Error reason: Both Data size and sequence ID without required attributes.

enum _xspi_exclusive_access_lock_mode

Exclusive access lock.

Values:

enumerator kXSPI_ExclusiveAccessLockDisabled

Write permissions available for all masters based on their MDxACP evaluation.

enumerator kXSPI_ExclusiveAccessLockEnabled

Write permissions revoked for all domains, Any write transaction coming to this flash address will result in an error..

enumerator kXSPI_ExclusiveAccessLockExceptMasterId

Write permissions are revoked for all masters except the master ID specified by FRADn_WORD2[EALO] fields.

enum _xspi_secure_attribute

XSPI defines the secure attribute selection criteria for entry into descriptor queue...

Values:

enumerator kXSPI_AttributeMasterNonsecureOnly

Allow the bus attribute for this master to non-secure only

enumerator kXSPI_AttributeMasterSecureOnly

Allow the bus attribute for this master to secure only

enumerator kXSPI_AttributeMasterNonsecureSecureBoth

Allow the bus master's attribute: Both secure and non-secure

enum _xspi_descriptor_lock

Descriptor lock.

Values:

enumerator kXSPI_DescriptorLockDisabled

Descriptor registers can be written by any master

enumerator kXSPI_DescriptorLockEnabledTillHardReset

FRAD descriptor is read-only till next hard reset.

enumerator kXSPI_DescriptorLockEnabledExceptMasterId

Descriptors are read-only. MDnACP fields can be programmed only by the master with ID matching the MID.

enumerator kXSPI_DescriptorLockEnabled
Descriptor registers are read-only.

enum _xspi_target_group
The enumeration of target group.
Values:

enumerator kXSPI_TargetGroup0
Target groupe queue 0

enumerator kXSPI_TargetGroup1
Target groupe queue 1

enum _xspi_subbuffer_division_factor
Subbuffer division factor.
Values:

enumerator kXSPI_Division0
division factor 0

enumerator kXSPI_Division2
division factor 2 subbuffer size/2

enumerator kXSPI_Division4
division factor 4 subbuffer size/4

enumerator kXSPI_Division8
division factor 8 subbuffer size/8

enumerator kXSPI_Division16
division factor 16 subbuffer size/16

enum _xspi_ahb_sub_buffer_status
The enumeration of sub buffer status, empty, full, or partially full.
Values:

enumerator kXSPI_AhbSubBufferEmpty
Current sub buffer is empty.

enumerator kXSPI_AhbSubBufferFull
Current sub buffer is full.

enumerator kXSPI_AhbSubBufferPartiallyFull
Current sub buffer contains some entries.

enum _xspi_ppw_flag_clear_policy
The enumeration of XSPI page program waiting flag clear policy.
Values:

enumerator kXSPI_SoftwareClearPPWFlag

Application should clear the flag manually, after check the WIP of external flag is cleared.

enumerator kXSPI_HardwareClearPPWFlag

The flag is cleared by hardware automatically, the variable in type of *xspi_device_status_reg_info_t* should be defined.

enum _xspi_ahb_split_size

XSPI AHB Split Size These bits are used to control the split size when split function is enabled BFGENCR[SPLITEN] is '1'.

Values:

enumerator kXSPI_AhbSplitSize8b

Enable AHB transaction split and split size is 8 bytes.

enumerator kXSPI_AhbSplitSize16b

Enable AHB transaction split and split size is 16 bytes.

enumerator kXSPI_AhbSplitSize32b

Enable AHB transaction split and split size is 32 bytes.

enumerator kXSPI_AhbSplitSize64b

Enable AHB transaction split and split size is 64 bytes.

enumerator kXSPI_AhbSplitSizeDisabled

Disable AHB Transaction Split feature.

enum _xspi_ahb_alignment

XSPI AHB access towards flash is broken if this AHB alignment boundary is crossed. This field is only supported XSPI0 XSPI1.

Values:

enumerator kXSPI_AhbAlignmentNoLimit

no limit

enumerator kXSPI_AhbAlignment256BLimit

256B limit

enumerator kXSPI_AhbAlignment512BLimit

512B limit

enumerator kXSPI_AhbAlignment1KBLimit

1KB limit

enum _xspi_ahb_request_suspend_state

The enumeration of AHB request suspend state.

Values:

enumerator kXSPI_AhbRequestNoSuspended
None of AHB request is suspended.

enumerator kXSPI_AhbRequestSuspended
The AHB request is suspended.

enumerator kXSPI_AhbRequestNotApplicable
The AHB request is not applicable.

enumerator kXSPI_AhbReqestResumed
The AHB request is resumed.

enum _xspi_otfad_prefetch_boundary
The enumeration of OTFAD prefetch address boundary.
Values:

enumerator kXSPI_OTFADPrefetchNoBoundary
No prefetch address boundary.

enumerator kXSPI_OTFADPrefetchBoundary1K
Prefetch address boundary is 1K.

enum _xspi_byte_order
Byte ordering endianness.
Values:

enumerator kXSPI_64BitBE
64 bit big endian

enumerator kXSPI_32BitLE
32 bit little endian

enumerator kXSPI_32BitBE
32 bit big endian

enumerator kXSPI_64BitLE
64 bit little endian

enum _xspi_cache64_policy
The enumeration of cache64 policy.
Values:

enumerator kXSPI_Cache64PolicyNonCacheable
Non-cacheable.

enumerator kXSPI_Cache64PolicyWriteThrough
Write through.

enumerator kXSPI_Cache64PolicyWriteBack

Write back.

typedef enum *_xspi_lut_instr* xspi_lut_instr_t

CMD definition of XSPI, use to form LUT instruction, xspi_lut_instr_t.

typedef enum *_xspi_pad* xspi_pad_t

pad definition of XSPI, use to form LUT instruction.

typedef enum *_xspi_flags* xspi_flags_t

XSPI interrupt status flags.

Deprecated:

For Error flag please use *xspi_error_flag_t* as instead, for cmd execution and arbitration flag please use *xspi_cmd_execution_arbitration_flag_t* as instead.

typedef struct *_xspi_device_status_reg_info* xspi_device_status_reg_info_t

The structure of external device's status register information(location, priority).

Note: Only useful when AHB write to flash device is enabled, and use hardware to clear ppw flag.

typedef enum *_xspi_dll_mode* xspi_dll_mode_t

Dll Mode enumerations.

typedef union *_xspi_dll_para* xspi_dll_para_t

The union of user custommed DLL parameters, in case of *kXSPI_BypassMode* bypassModePara is used, in case of *kXSPI_AutoUpdateMode* autoUpdateModoPara is used.

typedef struct *_xspi_dll_config* xspi_dll_config_t

The structure of Delay-lock-loop configuration.

typedef enum *_xspi_sample_clk_source* xspi_sample_clk_source_t

The enumeration of sample clock source which is used to sample data from external devices.

Note: If external device support Data strobe signal, the sample clock should selected as kXSPI_SampleClkFromExternalDQS.

typedef enum *_xspi_data_learning_pad_select* xspi_data_learning_pad_select_t

The selection of data learning pad.

typedef struct *_xspi_data_learning_config* xspi_data_learning_config_t

The configuration of data learning.

Note: Data learning feature only supported in the DQS sampling method(kXSPI_SampleClkFromExternalDQS).

typedef struct *_xspi_sample_clk_config* xspi_sample_clk_config_t

Sample clock configuration, which used to sample data from external device.

typedef enum *_xspi_device_addr_mode* xspi_device_addr_mode_t

External device's address mode.

typedef enum *_xspi_device_interface_type* xspi_device_interface_type_t

The enumeration of external device's interface type, hyper bus or SPI(including strandard and extended).

Note: Different interface result different settings.

typedef enum *_xspi_hyper_bus_x16_mode* xspi_hyper_bus_x16_mode_t

The enumeration of hyper bus X16 mode.

typedef union *_xspi_device_interface_settings* xspi_device_interface_settings_t

The union of external device's settings, if external device following hyper bus protocol #hyperBusSettings take effects, if external device following SPI protocol #strandardExtendedSPISettings take effects.

typedef enum *_xspi_ddr_data_aligned_clk* xspi_ddr_data_aligned_clk_t

Enumeration defining the alignment modes of XSPI DDR data with respect to the internal reference clock.

typedef struct *_xspi_device_ddr_config* xspi_device_ddr_config_t

The structure of device DDR configuration.

typedef struct *_xspi_device_config* xspi_device_config_t

External device configuration items.

typedef enum *_xspi_command_type* xspi_command_type_t

Command type.

typedef enum *_xspi_mdad_mask_type* xspi_mdad_mask_type_t

The enumeration of SFP MDAD mask type.

typedef enum *_xspi_mdad_error_reason* xspi_mdad_error_reason_t

The enumeration of MDAD error reasons.

typedef enum *_xspi_exclusive_access_lock_mode* xspi_exclusive_access_lock_mode_t

Exclusive access lock.

typedef enum *_xspi_secure_attribute* xspi_secure_attribute_t

XSPI defines the secure attribute selection criteria for entry into descriptor queue...

typedef struct *_xspi_mdad_config* xspi_mdad_config_t

MDAD configuration.

typedef struct *_xspi_sfp_mdad_config* xspi_sfp_mdad_config_t

The structure of SFP MDAD configurations for all target groups.

typedef enum *_xspi_descriptor_lock* xspi_descriptor_lock_t

Descriptor lock.

typedef struct *_xspi_frad_tg_config* xspi_frad_tg_config_t

FRAD configuration.

typedef struct *_xspi_frad_config_t* xspi_frad_config_t

FRAD configuration.

typedef struct *_xspi_sfp_frad_config* xspi_sfp_frad_config_t

The structure of SFP FRAD configurations.

typedef struct *_xspi_frad_transaction_info* xspi_frad_transaction_info_t

The structure of latest FRAD transaction information.

typedef struct *_xspi_ip_access_config* xspi_ip_access_config_t

Configurations of IP access(including IP read and write).

typedef enum *_xspi_target_group* xspi_target_group_t

The enumeration of target group.

typedef struct *_xspi_tg_add_write_status* xspi_tg_add_write_status_t

The status of latest target group address write operation.

typedef struct *_xspi_transfer* xspi_transfer_t

Transfer structure used for XSPI functional interface.

typedef struct *_xspi_handle* xspi_handle_t

typedef void (*xspi_transfer_callback_t)(XSPI_Type *base, *xspi_handle_t* *handle, *status_t* status, void *userData)

XSPI transfer callback function.

typedef enum *_xspi_subbuffer_division_factor* xspi_subbuffer_division_factor_t

Subbuffer division factor.

typedef struct *_xspi_ahbBuffer_sub_buffer_config* xspi_ahbBuffer_sub_buffer_config_t

The structure of sub-buffer configurations.

typedef struct *_xspi_ahbBuffer_config* xspi_ahbBuffer_config_t

Structure of AHB buffer configurations.

Note: When an AHB read access comes it is assigned to a buffer based on its master ID. Then it further assigned to a sub-buffer based on which sub-buffer address range this transaction lies into.

Note: When sub-buffer division is hit, in case of a buffer miss the controller will fetch the data equal to the size of that sub-buffer.

Note: If sub-buffer division is enabled for a buffer the hit/miss check is done at sub-buffer level. And if prefetch is enable, the prefetch takes place for individual sub-buffers.

typedef enum *_xspi_ahb_sub_buffer_status* xspi_ahb_sub_buffer_status_t

The enumeration of sub buffer status, empty, full, or partially full.

typedef struct *_xspi_ahbBuffer_perf_monitor_result* xspi_ahbBuffer_perf_monitor_result_t

The result of AHB buffer performance monitor, including buffer miss count and buffer hit count.

typedef enum *_xspi_ppw_flag_clear_policy* xspi_ppw_flag_clear_policy_t

The enumeration of XSPI page program waiting flag clear policy.

typedef struct *_xspi_ahb_write_config* xspi_ahb_write_config_t

The configuration of AHB write access.

typedef enum *_xspi_ahb_split_size* xspi_ahb_split_size_t

XSPI AHB Split Size These bits are used to control the split size when split function is enabled BFGENCR[SPLITEN] is '1'.

typedef enum *_xspi_ahb_alignment* xspi_ahb_alignment_t

XSPI AHB access towards flash is broken if this AHB alignment boundary is crossed. This field is only supported XSPI0 XSPI1.

typedef struct *_xspi_ahb_access_config* xspi_ahb_access_config_t

The structure of AHB access configurations, including AHB buffer settings, AHB Read Seq ID, AHB write settings.

typedef struct *_xspi_ahb_read_error_info* xspi_ahb_read_error_info_t

The structure of information when AHB read error occur.

typedef enum *_xspi_ahb_request_suspend_state* xspi_ahb_request_suspend_state_t

The enumeration of AHB request suspend state.

typedef struct *_xspi_ahb_request_suspend_info* xspi_ahb_request_suspend_info_t

The structure of information when AHB reqest is suspended.

typedef enum *_xspi_otfad_prefetch_boundary* xspi_otfad_prefetch_boundary_t

The enumeration of OTFAD prefetch address boundary.

typedef enum *_xspi_byte_order* xspi_byte_order_t

Byte ordering endianness.

typedef struct *_xspi_config* xspi_config_t

XSPI configuration structure.

typedef enum *_xspi_cache64_policy* xspi_cache64_policy_t

The enumeration of cache64 policy.

typedef struct *_xspi_cache64_config* xspi_cache64_region_config_t

The structure of cache64 configuration.

Region0 range from base address to region0TopAddr[28:10] + 0x3FF, region1 range from region0TopAddr[28:10] + 0x400 to region1TopAddr[28:10] + 0x3FF, region1TopAddr[28:10] + 0x400 to end address.

FSL_XSPI_DRIVER_VERSION

XSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)

Formula to form XSPI instructions in LUT table.

XSPI_TARGET_GROUP_COUNT

XSPI_SFP_FRAD_COUNT

XSPI_TG_REG_ADDR(base, tg, reg)

XSPI_TG_REG_VAL(base, tg, reg)

XSPI_TG_REG_ADDR_RBDR(base, tg)

XSPI_TG_REG_VAL_RBDR_INDEX(base, tg, i)

XSPI_TG_REG_ADDR_INT_EN(base, tg)

XSPI_TG_REG_ADDR_RSER(base, tg)

XSPI_IP_RX_BUFFER_SIZE

XSPI_IP_TX_BUFFER_SIZE

uint32_t XSPI_GetInstance(XSPI_Type *base)

Get the instance number for XSPI.

Parameters

- base – XSPI base pointer.

status_t XSPI_CheckAndClearErrorTG(XSPI_Type *base, *xspi_target_group_t* tgId, uint32_t status)

Check and clear IP command execution errors.

Parameters

- base – XSPI base pointer.
- tgId – Specify the target group.
- status – interrupt status.

void XSPI_Init(XSPI_Type *base, const *xspi_config_t* *ptrConfig)

Initializes the XSPI module and internal state.

This function configures the XSPI with the input configure parameters. Users should call this function before any XSPI operations.

Parameters

- base – XSPI peripheral base address.
- ptrConfig – XSPI configure structure.

void XSPI_GetDefaultConfig(*xspi_config_t* *ptrConfig)

Gets default settings for XSPI.

Warning: The parameter **devconfig** will be removed in next release.

Parameters

- base – XSPI peripheral base address.
- ptrConfig – XSPI configuration structure.

void XSPI_Deinit(XSPI_Type *base)

Deinitialize the XSPI module.

Clears the XSPI state and XSPI module registers.

Parameters

- base – XSPI peripheral base address.
- devconfig – Flash configuration parameters.

void XSPI_UpdateLUT(XSPI_Type *base, uint8_t index, const uint32_t *cmd, uint8_t count)

Updates the LUT table.

Parameters

- base – XSPI peripheral base address.
- index – From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 10 instructions and occupy 5*32-bit memory.
- cmd – Command sequence array.
- count – Number of instruction-operand pairs

static inline void XSPI_UpdateByteOrder(XSPI_Type *base, *xspi_byte_order_t* byteOrder)

static inline void XSPI_EnableModule(XSPI_Type *base, bool enable)

Enable or disable the XSPI module.

Parameters

- base – XSPI peripheral base address.
- enable – true means enable XSPI, false means disable.

static inline bool XSPI_CheckModuleEnabled(XSPI_Type *base)

Check if the XSPI module is enabled.

Parameters

- base – XSPI peripheral base address.

Return values

- true – XSPI module is enabled.
- false – XSPI module is disabled.

void XSPI_ResetSfmAndAhbDomain(XSPI_Type *base)

Reset Serial flash memory domain and AHB domain at the same time.

Note: Resetting only the Serial flash memory domain and AHB domain may cause undesirable side effects.

Parameters

- base – XSPI peripheral base address.

static inline void XSPI_ResetTgQueueTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Reset IPS target group queue.

Parameters

- base – XSPI peripheral base address.
- tgId – Target group ID.

static inline void XSPI_SoftwareReset(XSPI_Type *base)

Software reset flash memory domain, AHB domain, and Target group at the same time.

Parameters

- base – XSPI peripheral base address.

static inline bool XSPI_CheckGlobalConfigLocked(XSPI_Type *base)

Check if write access to registers(MGC, MRC, MTO, SFP_ARB_TIMEOUT, SFP_LUT_ENn, LUTn, BFGENCR) is locked.

Parameters

- base – XSPI peripheral base address.

Return values

- false – Write access to listed registers is not locked.
- true – Write access to listed registers is locked.

void XSPI_SetHyperBusX16Mode(XSPI_Type *base, *xspi_hyper_bus_x16_mode_t* x16Mode)

Set Hyper bus X16 mode.

Parameters

- base – XSPI peripheral base address.
- x16Mode – Specify X16 mode.

status_t XSPI_UpdateDeviceAddrMode(XSPI_Type *base, *xspi_device_addr_mode_t* addrMode)

Update address mode to access external device.

Parameters

- base – XSPI peripheral base address.
- addrMode – Specify the address mode to update.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail to update address mode due to AHB read access asserted.
- kStatus_XSPI_IPAccessAsserted – Fail to update address mode due to a IP access already asserted.
- kStatus_XSPI_AhbWriteAccessAsserted – Fail to update address mode due to AHB write access asserted.

- kStatus_Success – Successfully to update address mode.

static inline void XSPI_EnableVariableLatency(XSPI_Type *base, bool enable)

Enable/disable variable latency for XSPI.

Note: Only useful for hyper bus device.

Note: It is application responsibility to enable variable latency for external hyper bus device(by change device's register).

Parameters

- base – XSPI peripheral base address.
- enable – Specify operation:
  - **true** Enable variable latency;
  - **false** Disable variable latency.

static inline void XSPI_EnableDozeMode(XSPI_Type *base, bool enable)

Enable/disable Doze mode for XSPI controller.

Note: Once doze mode is enabled, the XSPI controller will enter disable mode when system in low power modes.

Parameters

- base – XSPI peripheral base address
- enable – Specify operation:
  - **true** Enable Doze mode;
  - **false** Disable Doze mode.

static inline void XSPI_SetSignalOutputValue(XSPI_Type *base, uint32_t signalMask, bool outputLogic)

Specify the logic level of the XSPI IOFA[3] and IOFA[2] output in the inactive state.

Parameters

- base – XSPI peripheral base address
- signalMask – Should be the OR'ed value of _xspi_external_signal.
- outputLogic – Output logic of selected signals,
  - **false** Output logic 0;
  - **true** Output logic 1.

static inline void XSPI_EnableInvertedSerialClockOutput(XSPI_Type *base, bool enable)

Enable/disable inverted serial clock output for XSPI controller.

Note: It is application responsibility to enable CLKn for external device(by change device's register).

Parameters

- base – **[in]** XSPI peripheral base address
- enable – **[in]** Enable/disable inverted serial clock output:
  - **true** Enable inverted serial clock output;
  - **false** Disable inverted serial clock output.

void XSPI_UpdateDllValue(XSPI_Type *base, *xspi_dll_config_t* *ptrDllConfig, bool enableDDR, bool enableX16Mode, uint32_t xspiRootClk)

Update DLL configuration.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrDllConfig – **[in]** Pointer to the DLL configuration.
- enableDDR – **[in]** DDR mode is enabled or not.
- enableX16Mode – **[in]** X16 mode is enabled or not.
- xspiRootClk – **[in]** The frequency of xspi root clock, the unit is Hz.

status_t XSPI_SetDataLearningConfig(XSPI_Type *base, *xspi_data_learning_config_t* *ptrDataLearningConfig)

Set Data learning configurations.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrDataLearningConfig – **[in]** Pointer to data learning configuration.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail to set data learning configuration due to AHB read access asserted.
- kStatus_XSPI_IPAccessAsserted – Fail to set data learning configuration due to a IP access already asserted.
- kStatus_XSPI_AhbWriteAccessAsserted – Fail to set data learning configuration due to AHB write access asserted.
- kStatus_Success – Successfully to set Data learning configurations.

static inline bool XSPI_CheckDataLearningFailure(XSPI_Type *base)

Check if data learning failure has detected.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – Data learning has failed.
- false – Data learning not fail.

static inline void XSPI_GetDataLearningEdgeMatchSignature(XSPI_Type *base, uint8_t *posEdgeMatch, uint8_t *negEdgeMatch)

Get data learning positive and negative edge match signature.

Parameters

- base – **[in]** XSPI peripheral base address.
- posEdgeMatch – **[out]** Pointer to the memory to store positive edge match signature.

- negEdgeMatch – **[out]** Pointer to the memory to store negative edge match signature.

status_t XSPI_SetDeviceConfig(XSPI_Type *base, *xspi_device_config_t* *devConfig)

Set device configuration.

Parameters

- base – **[in]** XSPI peripheral base address.
- devConfig – **[in]** Pointer to device configuration.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail to set device configuration due to AHB read access asserted.
- kStatus_XSPI_IPAccessAsserted – Fail to set device configuration due to a IP access already asserted.
- kStatus_XSPI_AhbWriteAccessAsserted – Fail to set device configuration due to AHB write access asserted.
- kStatus_Success – Successfully to set device configurations.

static inline void XSPI_EnableInterruptsTG(XSPI_Type *base, *xspi_target_group_t* tgId, uint64_t mask)

Enables the XSPI interrupts for specific target group.

Parameters

- base – XSPI peripheral base address.
- tgId – Target group ID.
- mask – XSPI interrupt source mask.

static inline void XSPI_DisableInterruptsTG(XSPI_Type *base, *xspi_target_group_t* tgId, uint64_t mask)

Disables the XSPI interrupts for specific target group.

Parameters

- base – XSPI peripheral base address.
- tgId – Target group ID.
- mask – XSPI interrupt source mask.

static inline void XSPI_EnableTxDMATG(XSPI_Type *base, *xspi_target_group_t* tgId, bool enable)

Enables or disables XSPI IP Tx FIFO DMA requests for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.
- enable – **[in]** Enable flag for transmit DMA request. Pass true for enable, false for disable.

static inline void XSPI_EnableRxDMATG(XSPI_Type *base, *xspi_target_group_t* tgId, bool enable)

Enables or disables XSPI IP Rx FIFO DMA requests for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

- enable – **[in]** Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t XSPI_GetTxFifoAddressTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Gets XSPI IP tx fifo address for DMA transfer for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Returns

The tx fifo address for the target group.

static inline uint32_t XSPI_GetRxFifoAddressTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Gets XSPI IP rx fifo address for DMA transfer for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Returns

The rx fifo address for the target group.

static inline uint32_t XSPI_GetErrorStatusFlagsTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Get error status flags for specific target group.

Parameters

- base – XSPI peripheral base address.
- tgId – Target group ID.

Returns

All asserted error status flags for the target group. Should be the OR'ed value of *xspi_error_flag_t*.

static inline void XSPI_ClearErrorStatusFlagsTG(XSPI_Type *base, *xspi_target_group_t* tgId, uint32_t flags)

Clear input error status flags for specific target group.

Parameters

- base – XSPI peripheral base address.
- tgId – Target group ID.
- flags – Error flags to clear. Should be the OR'ed value of *xspi_error_flag_t*.

static inline bool XSPI_CheckIpRequestGrantedTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Check if the IP access request was granted access for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Return values

- true – The IP access is granted arbitration for the target group.
- false – No IP access is queued for the target group.

static inline bool XSPI_GetBusIdleStatusTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Return whether the bus is idle for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Return values

- true – Bus is idle for the target group.
- false – Bus is busy for the target group.

static inline uint32_t XSPI_GetCmdExecutionArbitrationStatusFlagsTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Get asserted flags about SFM command execution and arbitration for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Returns

The assert flags about SFM command execution and arbitration for the target group. should be the OR'ed value of *xspi_cmd_execution_arbitration_flag_t*.

static inline void XSPI_ClearCmdExecutionArbitrationStatusFlagsTG(XSPI_Type *base, *xspi_target_group_t* tgId, uint32_t flags)

Clear asserted flags about SFM command execution and arbitration for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.
- flags – **[in]** The mask of flags to clear.

static inline bool XSPI_GetBusIdleStatus(XSPI_Type *base)

Return whether the bus is idle.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – Bus is idle.
- false – Bus is busy.

static inline bool XSPI_CheckAhbReadAccessAsserted(XSPI_Type *base)

Check if AHB read access has been requested or is ongoing.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – AHB read access is requested or is ongoing.
- false – AHB read access is not requested and is not ongoing.

static inline bool XSPI_CheckAhbWriteAccessAsserted(XSPI_Type *base)

Check if AHB write access has been requested or is ongoing.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – AHB read access is requested or is ongoing.
- false – AHB read access is not requested and is not ongoing.

static inline void XSPI_ResetTxRxBufferTG(XSPI_Type *base, *xspi_target_group_t* tgId, bool txFifo, bool rxFifo)

Clear the XSPI IP TX/RX buffer logic.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.
- txFifo – **[in]** Pass true to reset TX FIFO.
- rxFifo – **[in]** Pass true to reset RX FIFO.

static inline void XSPI_ClearTxBufferTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Clear TX buffer for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

static inline bool XSPI_CheckTxBuffLockOpenTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Check if IP manager can write to TX buffer for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Return values

- true – Tx buffer lock is open.
- false – Tx buffer lock is not open.

static inline void XSPI_WriteTxBufferTG(XSPI_Type *base, *xspi_target_group_t* tgId, uint32_t data)

Writes data into IPS TX Buffer for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address
- tgId – **[in]** Target group ID.
- data – **[in]** The data bytes to send.

status_t XSPI_UpdateTxBufferWaterMarkTG(XSPI_Type *base, *xspi_target_group_t* tgId, uint32_t waterMark)

Update watermark for TX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.
- waterMark – **[in]** The watermark to set, the unit is byte, should be the multiple of 4 byte.

Return values

- kStatus_XSPI_IPAccessAsserted – Fail to update watermark for Tx buffer, due to IP access is asserted.

- kStatus_XSPI_WaterMarkIllegal – Fail to update watermark for Tx buffer, due to input watermark is not the multiple of 4 bytes.
- kStatus_Success – Successful to update watermark.

static inline void XSPI_ClearRxBufferTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Clear RX buffer for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

static inline uint32_t XSPI_ReadRxBufferTG(XSPI_Type *base, *xspi_target_group_t* tgId, uint8_t fifoIndex)

Receive data from IPX RX FIFO for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address
- tgId – **[in]** Target group ID.
- fifoIndex – **[in]** Source fifo index.

Returns

The data in the FIFO.

static inline void XSPI_TriggerRxBufferPopEventTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Trigger a pop event for RX buffer pop event for specific target group.

Note: Each pop event discard watermark + 1 enties from RX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

static inline bool XSPI_CheckRxBufferWaterMarkExceedTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Check if RX buffer watermark is exceed for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Return values

- true – The RX buffer watermark has been excceded.
- false – The RX buffer watermark has not been exceeded.

static inline uint32_t XSPI_GetRxBufferAvailableBytesCountTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Get RX buffer aviailable bytes count for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Returns

The available counts if bytes in RX buffer.

static inline uint32_t XSPI_GetRxBufferRemovedBytesCountTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Get counts of bytes already removed from RX buffer for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Returns

Counts of removed bytes.

static inline bool XSPI_CheckIpWriteTriggeredTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Check if IP write access is triggered for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Return values

- true – The IP write access is granted arbitration.
- false – No IP write access is queued.

static inline bool XSPI_CheckIpReadTriggeredTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Check if IP read access is triggered for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Return values

- true – The IP read access is granted arbitration.
- false – No IP read access is queued.

static inline bool XSPI_CheckFSMValidTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Check if IPS transfer is granted arbitration or execution for specific target group.

Note: The FSMSTAT[VLD] goes to 0 once the IPS transfer is completed and XSPI is IDLE.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Return values

- true – Valid, the IPS transfer is granted arbitration or execution.
- false – Not valid, no IPS transfer is queued.

static inline bool XSPI_CheckIPAccessAssertedTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Check if IP access is asserted for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Target group ID.

Return values

- false – The Access triggered by IP bus is not asserted.
- true – The Access triggered by IP bus is asserted.

static inline bool XSPI_CheckIPAccessAsserted(XSPI_Type *base)

Check if IP access is asserted.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- false – The Access triggered by IP bus is not asserted.
- true – The Access triggered by IP bus is asserted.

status_t XSPI_UpdateRxBufferWaterMarkTG(XSPI_Type *base, *xspi_target_group_t* tgId, uint32_t waterMark)

Update watermark for RX buffer.

```
Set watermark as 4 bytes:
XSPI_UpdateRxBufferWaterMarkTG(XSPI0, tgId, 4UL);
Set watermark as 8 bytes:
XSPI_UpdateRxBufferWaterMarkTG(XSPI0, tgId, 8UL);
```

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Specify the target group.
- waterMark – **[in]** Specify the number of bytes in the RX buffer which causes XSPI to assert the watermark exceeded flag, should be in multiple of 4 bytes.

Return values

- kStatus_XSPI_IPAccessAsserted – Fail to update watermark for Rx buffer, due to IP access is asserted.
- kStatus_XSPI_WaterMarkIllegal – Fail to update watermark for Tx buffer, due to input watermark is not the multiple of 4 bytes.
- kStatus_Success – Successful to update watermark.

void XSPI_SetSFPFradEALModeTG(XSPI_Type *base, *xspi_target_group_t* tgId, *xspi_exclusive_access_lock_mode_t* ealMode, uint8_t fradId)

Set exclusive access lock mode for the specific frad.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Specify the target group.
- ealMode – **[in]** Specify the exclusive access lock mode.
- fradId – **[in]** Specify the frad.

void XSPI_GetFradLastTransactionsInfo(XSPI_Type *base, *xspi_target_group_t* tgId, *xspi_frad_transaction_info_t* *ptrInfo, uint8_t fradId)

Get FARD latest transaction information.

Parameters

- base – **[in]** XSPI peripheral base address.

- tgId – **[in]** Specify the target group.
- ptrInfo – **[out]** Pointer to the variable in type of *xspi_frad_transaction_info_t* to store information.
- fradId – **[in]** Specify the frad Id.

void XSPI_UpdateSFPConfig(XSPI_Type *base, *xspi_sfp_mdad_config_t* *ptrSfpMdadConfig, *xspi_sfp_frad_config_t* *ptrSfpFradConfig)

Update SFP configurations, including MDAD configurations and FRAD configurations.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrMdadConfig – **[in]** Pointer to the SFP MDAD configuration.
- ptrFradConfig – **[in]** Pointer to the SFP FRAD configuration.

static inline bool XSPI_CheckSFPFradEnabled(XSPI_Type *base)

Check if SFP FRAD check is enabled for IP write access.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- false – SFP FRAD check is disabled.
- true – SFP FRAD check is enabled for IP write access.

xspi_mdad_error_reason_t XSPI_GetMdadErrorReason(XSPI_Type *base, *xspi_target_group_t* tgId)

Get MDAD check error reason for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Specify the target group.

Returns

The details of MDAD error reason, in type of *xspi_mdad_error_reason_t*.

status_t XSPI_UpdateSFPArbitrationLockTimeoutCounter(XSPI_Type *base, uint32_t countValue)

Update SFP arbitration lock timeout counter.

Note: The SFP arbitration lock time out counter starts when Page program wait flag asserted.

Parameters

- base – **[in]** XSPI peripheral base address.
- countValue – **[in]** The count value, specify the time in IPS clock cycles.

Return values

- kStatus_XSPI_RegWriteLocked – Write operation to related register is locked.
- kStatus_Success – Success to update timeout counter.

void XSPI_ClearTgAddrWriteStatus(XSPI_Type *base, *xspi_target_group_t* tgId)

Clear address write status for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Specify the target group to clear address write status.

void XSPI_GetTgAddrWriteStatus(XSPI_Type *base, *xspi_target_group_t* tgId, *xspi_tg_add_write_status_t* *ptrStatus)

Get address write status for specific target group.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Specify the target group.
- ptrStatus – **[out]** Pointer to the variable in type of *xspi_tg_add_write_status_t* to store address write status.

void XSPI_UnlockIpAccessArbitrationTG(XSPI_Type *base, *xspi_target_group_t* tgId)

Unlock Ip access arbitration.

Parameters

- base – **[in]** XSPI peripheral base address.
- tgId – **[in]** Specify the target group.

static inline void XSPI_ClearIPAccessSeqPointer(XSPI_Type *base)

Clear Ip access sequence pointer.

Parameters

- base – **[in]** XSPI peripheral base address.

status_t XSPI_UpdateIPAccessTimeoutCounter(XSPI_Type *base, uint32_t countValue)

Update the count of SFP access timeout counter.

Note: The counter starts when any IP access pass SFP check(if enabled), and request is granted by XSPI arbiter and XSPI is IDLE.

Note: The counter does not start in case of IP access was granted by XSPI is not IDLE.

Parameters

- base – XSPI peripheral base address.
- countValue – The count value, specify the time in IPS clock cycles.

Return values

- kStatus_XSPI_RegWriteLocked – Write operation to related register is locked.
- kStatus_Success – Success to update timeout counter.

status_t XSPI_StartIpAccess(XSPI_Type *base, uint32_t addr, uint8_t seqIndex, size_t byteSize, *xspi_target_group_t* tgId, bool lockArbitration)

Start IP access(including read and write) in blocking way.

Parameters

- base – **[in]** XSPI peripheral base address.
- addr – **[in]** Address of external device to read/write.
- seqIndex – **[in]** Sequence Id of the pre-programmed LUT.
- byteSize – **[in]** Size of data to read/write, the unit of byte.
- tgId – **[in]** Specify the target group used to write/read.
- lockArbitration – **[in]** Lock arbitration or not.

Return values

- kStatus_XSPI_IpAccessAddrSettingInvalid – Wrong Address input.
- kStatus_XSPI_IpAccessIPCRInvalid – Wrong seqindex or bytesize input.
- kStatus_Success – Success to start Ip access.

status_t XSPI_StartIpAccessNonBlocking(XSPI_Type *base, uint32_t addr, uint8_t seqIndex, size_t byteSize, *xspi_target_group_t* tgId, bool lockArbitration)

Start IP access in non-blocking way.

Parameters

- base – **[in]** XSPI peripheral base address.
- addr – **[in]** Address of external device to read/write.
- seqIndex – **[in]** Sequence Id of the pre-programmed LUT.
- byteSize – **[in]** Size of data to transfer, the unit of byte.
- tgId – **[in]** Specify the target group to use.
- lockArbitration – **[in]** Lock arbitration or not.

Return values

- kStatus_Success – Success to start Ip access.
- kStatus_XSPI_IpAccessIPCRInvalid – Invalid to set IPCR register.
- kStatus_XSPI_IpAccessAddrSettingInvalid – Invalid to set SFAR register.

status_t XSPI_SetIpAccessConfig(XSPI_Type *base, *xspi_ip_access_config_t* *ptrIpAccessConfig)

Set IP access configurations, including SFP configurations, sfp arbitration lock timeout value, Ip access timeout value.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrIpAccessConfig – **[in]** Pointer to the variable which contains Ip access configurations.

Return values

- kStatus_XSPI_RegWriteLocked – Write operation to related register is locked.
- kStatus_Success – Success to update timeout counter.

status_t XSPI_WriteBlockingTG(XSPI_Type *base, *xspi_target_group_t* tgId, uint8_t *buffer, size_t size)

Sends a buffer of data bytes using blocking method for specific target group.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – XSPI peripheral base address
- tgId – Target group ID
- buffer – The data bytes to send
- size – The number of data bytes to send

Return values

- kStatus_Success – write success without error
- kStatus_XSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_XSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_XSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t XSPI_ReadBlockingTG(XSPI_Type *base, *xspi_target_group_t* tgId, uint8_t *buffer, size_t size)

Receives a buffer of data bytes using a blocking method for specific target group.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – XSPI peripheral base address
- tgId – Target group ID
- buffer – The data bytes to send
- size – The number of data bytes to receive

Return values

- kStatus_Success – read success without error
- kStatus_XSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_XSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_XSPI_IpCommandGrantTimeout – IP command grant timeout detected

void XSPI_TransferCreateHandle(XSPI_Type *base, *xspi_handle_t* *handle, *xspi_transfer_callback_t* callback, void *userData, *xspi_target_group_t* targetGroup)

Initializes the XSPI handle for specific target group which is used in transactional functions.

Parameters

- base – XSPI peripheral base address.
- handle – pointer to xspi_handle_t structure to store the transfer state.
- callback – pointer to user callback function.
- userData – user parameter passed to the callback function.
- targetGroup – target group ID for this handle.

status_t XSPI_TransferBlocking(XSPI_Type *base, *xspi_transfer_t* *xfer)

Execute command to transfer a buffer data bytes using a blocking method.

Note: In case of flash page program, if WEL(Write enable) automatically cleared after program operation, then WEL should be asserted before invoking this API and the transfer size should not bigger than page size.

Parameters

- base – **[in]** XSPI peripheral base address
- xfer – **[in]** pointer to the transfer structure.

Return values

- kStatus_Success – command transfer success without error
- kStatus_XSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_XSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_XSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t XSPI_TransferNonBlocking(XSPI_Type *base, *xspi_handle_t* *handle, *xspi_transfer_t* *xfer)

Perform a interrupt non-blocking transfer on the XSPI bus.

Note: Calling the API returns immediately after transfer initiates. The user needs to call XSPI_GetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_XSPI_Busy, the transfer is finished. For XSPI_Read, the dataSize should be multiple of rx watermark level, or XSPI could not read data properly.

Parameters

- base – **[in]** XSPI peripheral base address.
- handle – **[in]** pointer to xspi_handle_t structure which stores the transfer state.
- xfer – **[in]** pointer to xspi_transfer_t structure.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_XSPI_Busy – Previous transmission still not finished.

status_t XSPI_TransferGetCount(XSPI_Type *base, *xspi_handle_t* *handle, size_t *count)

Get the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – **[in]** XSPI peripheral base address.
- handle – **[in]** pointer to xspi_handle_t structure which stores the transfer state.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

void XSPI_TransferAbort(XSPI_Type *base, *xspi_handle_t* *handle)

Abort an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – **[in]** XSPI peripheral base address.
- handle – **[in]** pointer to xspi_handle_t structure which stores the transfer state

static inline void XSPI_ClearAhbBuffer(XSPI_Type *base)

Blocks to clear AHB buffer and abort ongoing prefetch transaction(if prefetch is enabled.)

Note: The function return only after all AHB buffer pointers are cleared.

Parameters

- base – **[in]** XSPI peripheral base address.

status_t XSPI_EnableAhbBufferWriteFlush(XSPI_Type *base, bool enable)

Enable/disable the clearing of AHB read prefetch buffers when the same flash address is written by an AHB or IP command.

Parameters

- base – **[in]** XSPI peripheral base address.
- enable – **[in]** Used to enable/disable write flush.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted.
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted.
- kStatus_Success – Successfully to enable/disable AHB buffer write flush.

status_t XSPI_SetAhbBufferConfig(XSPI_Type *base, *xspi_ahbBuffer_config_t* *ptrBuffer0Config,
xspi_ahbBuffer_config_t *ptrBuffer1Config,
xspi_ahbBuffer_config_t *ptrBuffer2Config,
xspi_ahbBuffer_config_t *ptrBuffer3Config)

Set AHB buffer configurations.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrBuffer0Config – **[in]** Pointer to the variable which contain buffer0 configurations.
- ptrBuffer1Config – **[in]** Pointer to the variable which contain buffer1 configurations.

- ptrBuffer2Config – **[in]** Pointer to the variable which contain buffer2 configurations.
- ptrBuffer3Config – **[in]** Pointer to the variable which contain buffer3 configurations.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted.
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted.
- kStatus_Success – Success to set AHB buffer configurations.

status_t XSPI_UpdateAhbBufferSize(XSPI_Type *base, uint16_t buf0Size, uint16_t buf1Size, uint16_t buf2Size, uint16_t buf3Size)

Set Buffer size for all 4 AHB buffers.

Parameters

- base – **[in]** XSPI peripheral base address.
- buf0Size – **[in]** Specify size of AHB buffer0, range of 512, 256, 128, 64, 32, 16, 8, 4, 2, 0.
- buf1Size – **[in]** Specify size of AHB buffer1, range of 512, 256, 128, 64, 32, 16, 8, 4, 2, 0.
- buf2Size – **[in]** Specify size of AHB buffer2, range of 512, 256, 128, 64, 32, 16, 8, 4, 2, 0.
- buf3Size – **[in]** Specify size of AHB buffer3, range of 512, 256, 128, 64, 32, 16, 8, 4, 2, 0.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted.
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted.
- kStatus_Success – Success to set AHB buffer size.

xspi_ahb_sub_buffer_status_t XSPI_GetAhbSubBufferStatus(XSPI_Type *base, uint8_t ahbBufferId, uint8_t subBufferId)

Get status of AHB sub buffer.

Note: This function only useful when sub-division feature of the selected AHB buffer is enabled.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbBufferId – **[in]** The Id of AHB buffer, range from 0 to 3.
- subBufferId – **[in]** The Id of AHB buffer sub division, range from 0 to 3.

Returns

Current status of selected AHB sub buffer, in type of *xspi_ahb_sub_buffer_status_t*.

static inline void XSPI_StartAhbBufferPerfMonitor(XSPI_Type *base)

Start AHB buffer performance monitor.

Parameters

- base – **[in]** XSPI peripheral base address.

void XSPI_EnableAhbBufferPerfMonitor(XSPI_Type *base, uint8_t ahbBufferId, uint8_t subBufferId)

Enable AHB buffer performance monitor for selected AHB buffer's sub buffer.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbBufferId – **[in]** Specify the selected AHB buffer.
- subBufferId – **[in]** Specify the selected sub-buffer.

static inline void XSPI_DisableAhbBufferPerfMonitor(XSPI_Type *base, uint8_t ahbBufferId)

Disable AHB buffer performance monitor for selected AHB buffer.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbBufferId – **[in]** Specify the selected AHB buffer.

static inline void XSPI_StopAhbBufferPerfMonitor(XSPI_Type *base)

Stop AHB buffer performance monitor.

Parameters

- base – **[in]** XSPI peripheral base address.

static inline void XSPI_GetAhbBufferPerfMonitorResult(XSPI_Type *base, uint8_t ahbBufferId, *xspi_ahbBuffer_perf_monitor_result_t* *ptrPerfMonitorResult)

Get AHB buffer performance monitor result.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbBufferId – **[in]** Specify AHB buffer Id.
- ptrPerfMonitorResult – **[out]** Pointer to the variable to store selected AHB buffer's performance monitor result.

static inline bool XSPI_CheckAhbBufferPerfMonitorTimeCounterOverflow(XSPI_Type *base)

Check if AHB buffer performance monitor timeout counter is overflow.

Parameters

- base – XSPI peripheral base address.

Return values

- false – AHB buffer performance monitor timeout counter is not overflow.
- true – AHB buffer performance monitor timeout counter is overflow.

static inline bool XSPI_CheckAhbBufferPerfMonitorHitOverflow(XSPI_Type *base, uint8_t ahbBufferId)

Check if AHB buffer performance monitor buffer hit counter is overflow.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbBufferId – **[in]** Specify the AHB buffer Id.

Return values

- false – Overflow not detected.
- true – Overflow is detected.

static inline bool XSPI_CheckAhbBufferPerfMonitorMissOverflow(XSPI_Type *base, uint8_t ahbBufferId)

Check if AHB buffer performance monitor buffer miss counter is overflow.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbBufferId – **[in]** Specify the AHB buffer Id.

Return values

- false – Overflow not detected.
- true – Overflow is detected.

static inline uint32_t XSPI_GetAhbBufferPerfMonitorTimeCounter(XSPI_Type *base)

Get AHB buffer performance monitor timeout value.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

The value of time counter, in AHB clock cycles, since the performance monitor was running.

status_t XSPI_SetAhbAccessSplitSize(XSPI_Type *base, *xspi_ahb_split_size_t* ahbSplitSize)

Set AHB transaction(read/write) split size.

Note: For AHB write, if enable split feature(ahbSplitSize not set as kXSPI_AhbSplitSizeDisabled), XSPI will split one single AHB write burst into smaller bursts on external device side.

Note: For AHB read, if enable split feature, HW will realign the prefetch size to split size and in that way split a single read burst into smaller bursts.

Parameters

- base – **[in]** XSPI peripheral base address.
- ahbSplitSize – **[in]** Specify the AHB split size.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set AHB access split size.

status_t XSPI_SetAhbAccessBoundary(XSPI_Type *base, *xspi_ahb_alignment_t* alignment)

Set AHB transaction(read/write) boundary.

Note: For AHB write, XSPI keeps track of the start address and then compares it with the address of subsequent transaction receive on the AHB bus, when the input alignment is reached, XSPI negates AHB HREADY to block new accesses. This signal will be kept low until all the previous received write data is written to external memory and chip select is de-asserted. After that it allows next AHB write data to be received by making HREADY high thus ensuring that transaction is split at address boundary on external memory.

Note: For AHB read, XSPI checks the start address and end address to see if it is crossing the address boundary specified by input alignment. If the transaction crosses the address boundary, then it reduces the transaction size such that the data pre-fetch is stopped before the address boundary. Now if the AHB master is reading the data sequentially it will get buffer hits up to the input alignment boundary. When it reaches the next address boundary it will get a buffer miss and a new data pre-fetch will be launched towards the external memory device.

Parameters

- base – **[in]** XSPI peripheral base address.
- alignment – **[in]** Specify the AHB alignment, in type of *xspi_ahb_alignment_t*.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set AHB access boundary.

status_t XSPI_SetAhbReadDataSeqId(XSPI_Type *base, uint8_t seqId)

Set AHB read sequence Id.

Parameters

- base – **[in]** XSPI peripheral base address.
- seqId – **[in]** Specify the sequence Id in LUT used for AHB read.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set AHB read sequence Id.

status_t XSPI_SetAhbWriteDataSeqId(XSPI_Type *base, uint8_t seqId)

Set AHB write sequence Id.

Parameters

- base – **[in]** XSPI peripheral base address.
- seqId – **[in]** Specify the sequence Id in LUT used for AHB write.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set AHB write sequence Id.

status_t XSPI_UpdateAhbHreadyTimeOutValue(XSPI_Type *base, uint16_t timeoutValue)

Specify how long XSPI can hold HEADY low while waiting for response to an AHB transfer.

Note: If the specified time out value expired, an AHB illegal transaction error will be triggerred.

Parameters

- base – **[in]** XSPI peripheral base address.
- timeoutValue – **[in]** In multiples of 50000 AHB clock cycles, default value is 3, it means waiting for 150000 AHB clock cycles.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_Success – Success to set AHB HREADY timeout value.

status_t XSPI_SetAhbErrorPayload(XSPI_Type *base, uint32_t highPayload, uint32_t lowPayload)

Pre-define error payload which will be provided on the read data bus when the HEADY timeout counter expires.

Note: If the incoming AHB read request master is not mapped to any of the AHB buffer, the XSPI will keep the AHB HREADY signal low and will provided this pre-defined error payload on the AHB read data bus when the HEADY timeout counter expires.

Parameters

- base – **[in]** XSPI peripheral base address.
- highPayload – **[in]** High 32bits payload to set.
- lowPayload – **[in]** Low 32bits payload to set.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_Success – Success to set AHB HREADY timeout value.

xspi_ahb_read_error_info_t XSPI_ReturnAhbReadErrorInfo(XSPI_Type *base)

Return AHB read error information.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

Latest AHB read error information, in type of xspi_ahb_read_error_info_t.

static inline void XSPI_ClearAhbAccessSeqPointer(XSPI_Type *base)

Clear AHB access sequence pointer.

Parameters

- base – **[in]** XSPI peripheral base address.

void XSPI_GetAhbRequestSuspendInfo(XSPI_Type *base, *xspi_ahb_request_suspend_info_t* *ptrSuspendInfo)

Get Ahb request suspend information if priority mechanism is enabled.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrSuspendInfo – **[out]** Contain latest suspend info, the value may be 0xFF if the item is invalid in current suspend state.

static inline void XSPI_EnableAhbReadPrefetch(XSPI_Type *base, bool enable)

Enable/disable AHB Read Prefetch feature.

Parameters

- base – **[in]** XSPI peripheral base address.
- enable – **[in]** Used to enable/disable AHB read prefetch:
  - **false** Disable AHB read prefetch;
  - **true** Enable AHB read prefetch.

status_t XSPI_BlockAccessAfterAhbWrite(XSPI_Type *base, bool blockSequentWrite, bool blockRead)

Block access(write and read) after the AHB write operation.

Note: If external flash support RWW, read should not be blocked:

```
XSPI_BlockAccessAfterAhbWrite(XSPI0, true, false);
```

Note: If either of access is blocked, the page program wait flag will be asserted after an AHB write sequence completed. The assertion of flag will lock the arbitration and all access to the external memory are blocked. And the internal “page wait time” counter starts(Invoke XSPI_UpdatePageWaitTimeCounter to update counter value). After this counter reaches the value, a read is triggered by the XSPI module to read external device’s status register(The seq id should be pre-defined by XSPI_SetAhbReadStatusRegSeqId),

and the value is stored in the XSPI internal regsiter. And there are two options(Invoke XSPI_SelectPPWFlagClearPolicy to select) to clear the asserted page program wait flag.

a. Automatic cleared by XSPI hardware;

b. Cleared by software.

Note:

As soon as the page program wait flag is asserted, another counter(SFP arbitration

lock counter) also started, if the flag not cleared in desired counter value the arbitration lock timeout error will be generated.

Parameters

- base – **[in]** XSPI peripheral base address.
- blockSequentWrite – **[in]** Block sequence write or not.
- blockRead – **[in]** Block read or not.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_Success – Success to set related registers..

status_t XSPI_SelectPPWFlagClearPolicy(XSPI_Type *base, *xspi_ppw_flag_clear_policy_t* policy)

Set Page program wait flag clear policy.

Note: If set as hardware policy, the device's WIP information should be set by invoking *XSPI_SetSFMStatusRegInfo()*

Parameters

- base – **[in]** XSPI peripheral base address.
- policy – **[in]** Specify the policy to clear page program wait flag.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set PPW flag clear policy.

status_t XSPI_UpdatePageWaitTimeCounter(XSPI_Type *base, uint32_t countValue)

Update page wait timeout counter.

Parameters

- base – **[in]** XSPI peripheral base address.

- countValue – **[in]** The value of counter, in AHB clock cycles.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_PageProgramWaitFlagAsserted – Page program wait flag already asserted.
- kStatus_Success – Successfully to update page wait timeout counter.

status_t XSPI_SetAhbReadStatusRegSeqId(XSPI_Type *base, uint8_t seqId)

Set AHB read status register sequence Id.

Note: The read status sequence only triggerred when the page wait time counter expired.

Parameters

- base – **[in]** XSPI peripheral base address.
- seqId – **[in]** Specify the sequence Id in LUT used for AHB read status register.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set AHB read status register sequence Id.

static inline uint16_t XSPI_GetSFMStatusRegValue(XSPI_Type *base)

Get external serial flash's status register value.

Note: This function is only useful after the Page program wait flag is asserted due to either wait/read access is blocked after the AHB write operation.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

The status regsiter value of external device.

status_t XSPI_SetSFMStatusRegInfo(XSPI_Type *base, *xspi_device_status_reg_info_t* *ptrStatusRegInfo)

Set Serial flash memory status register information.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrStatusRegInfo – **[in]** Pointer to the variable which contain status register information.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_PageProgramWaitFlagAsserted – Page program wait flag already asserted.
- kStatus_Success – Successfully to set status register information.

status_t XSPI_EnableAhbWriteTerminate(XSPI_Type *base, bool enable)

Enable or disable AHB write terminate functionality.

Parameters

- base – XSPI peripheral base address.
- enable – True to enable AHB write terminate, false to disable.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_Success – Success to enable/disable AHB write terminate functionality.

status_t XSPI_SetAhbAccessConfig(XSPI_Type *base, *xspi_ahb_access_config_t* *ptrAhbAccessConfig)

Set AHB access configuration.

Parameters

- base – **[in]** XSPI peripheral base address.
- ptrAhbAccessConfig – **[in]** Pointer to the variable which contains AHB access configurations.

Return values

- kStatus_XSPI_AhbReadAccessAsserted – Fail due to an AHB read access already asserted
- kStatus_XSPI_AhbWriteAccessAsserted – Fail due to an AHB write access already asserted
- kStatus_XSPI_RegWriteLocked – Fail due to write operation to related registers is locked.
- kStatus_Success – Success to set AHB read status register sequence Id.

static inline void XSPI_EnableTxDMA(XSPI_Type *base, bool enable)

Enables or disables XSPI IP Tx FIFO DMA requests.

Parameters

- base – **[in]** XSPI peripheral base address.
- enable – **[in]** Enable flag for transmit DMA request. Pass true for enable, false for disable.

static inline void XSPI_EnableRxDMA(XSPI_Type *base, bool enable)

Enables or disables XSPI IP Rx FIFO DMA requests.

Parameters

- base – **[in]** XSPI peripheral base address.
- enable – **[in]** Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t XSPI_GetTxFifoAddress(XSPI_Type *base)

Gets XSPI IP tx fifo address for DMA transfer.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

The tx fifo address.

static inline uint32_t XSPI_GetRxFifoAddress(XSPI_Type *base)

Gets XSPI IP rx fifo address for DMA transfer.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

The rx fifo address.

static inline uint32_t XSPI_GetErrorStatusFlags(XSPI_Type *base)

Get error status flags.

Parameters

- base – XSPI peripheral base address.

Returns

All asserted error status flags for the target group. Should be the OR'ed value of *xspi_error_flag_t*.

static inline void XSPI_ClearErrorStatusFlags(XSPI_Type *base, uint32_t flags)

Clear input error status flags.

Parameters

- base – XSPI peripheral base address.
- flags – Error flags to clear. Should be the OR'ed value of *xspi_error_flag_t*.

static inline uint32_t XSPI_GetInterruptStatusFlags(XSPI_Type *base)

Get the XSPI interrupt status flags.

Deprecated:

Please use *XSPI_GetErrorStatusFlags()* as instead.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

interrupt status flag, use status flag to AND *xspi_flags_t* could get the related status.

static inline bool XSPI_CheckIpRequestGranted(XSPI_Type *base)

Check if the IP access request was granted access.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – IP request granted.
- false – IP request not granted.

static inline uint32_t XSPI_GetCmdExecutionArbitrationStatusFlags(XSPI_Type *base)

Get asserted flags about SFM command execution and arbitration.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

The assert flags about SFM command execution and arbitration, should be the OR'ed value of *xspi_cmd_execution_arbitration_flag_t*.

static inline void XSPI_ClearCmdExecutionArbitrationStatusFlags(XSPI_Type *base, uint32_t flags)

Clear asserted flags about SFM command execution and arbitration.

Parameters

- base – **[in]** XSPI peripheral base address.
- flags – **[in]** The mask of flags to clear.

static inline void XSPI_ClearTxBuffer(XSPI_Type *base)

Clear TX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.

static inline void XSPI_ClearRxBuffer(XSPI_Type *base)

Clear RX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.

static inline void XSPI_ResetTxRxBuffer(XSPI_Type *base, bool txFifo, bool rxFifo)

Clear the XSPI IP TX/RX buffer logic.

Parameters

- base – **[in]** XSPI peripheral base address.
- txFifo – **[in]** Pass true to reset TX FIFO.
- rxFifo – **[in]** Pass true to reset RX FIFO.

static inline *status_t* XSPI_UpdateRxBufferWaterMark(XSPI_Type *base, uint32_t waterMark)

Update watermark for RX buffer.

```
Set watermark as 4 bytes:
XSPI_UpdateRxBufferWaterMark(XSPI0, 4UL);
Set watermark as 8 bytes:
XSPI_UpdateRxBufferWaterMark(XSPI0, 8UL);
```

Parameters

- base – **[in]** XSPI peripheral base address.
- waterMark – **[in]** Specify the number of bytes in the RX buffer which causes XSPI to assert the watermark exceeded flag, should be in multiple of 4 bytes.

Return values

- kStatus_XSPI_IPAccessAsserted – Fail to update watermark for Rx buffer, due to IP access is asserted.
- kStatus_XSPI_WaterMarkIllegal – Fail to update watermark for Tx buffer, due to input watermark is not the multiple of 4 bytes.
- kStatus_Success – Successful to update watermark.

static inline *status_t* XSPI_UpdateTxBufferWaterMark(XSPI_Type *base, uint32_t waterMark)

Update watermark for TX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.
- waterMark – **[in]** The watermark to set.

Returns

Status of the operation.

static inline void XSPI_UnlockIpAccessArbitration(XSPI_Type *base)

Unlock IP access arbitration.

Parameters

- base – **[in]** XSPI peripheral base address.

static inline *status_t* XSPI_CheckAndClearError(XSPI_Type *base, uint32_t status)

Check and clear IP command execution errors.

Parameters

- base – XSPI peripheral base address.
- status – Interrupt status.

Returns

Status of the operation.

static inline void XSPI_WriteTxBuffer(XSPI_Type *base, uint32_t data)

Writes data into IPS TX Buffer.

Parameters

- base – **[in]** XSPI peripheral base address.
- data – **[in]** The data bytes to send.

static inline uint32_t XSPI_ReadRxBuffer(XSPI_Type *base, uint8_t fifoIndex)

Receive data from IPX RX FIFO.

Parameters

- base – **[in]** XSPI peripheral base address.
- fifoIndex – **[in]** Source fifo index.

Returns

The data in the FIFO.

static inline void XSPI_TriggerRxBufferPopEvent(XSPI_Type *base)

Trigger a pop event for RX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.

static inline bool XSPI_CheckRxBufferWaterMarkExceed(XSPI_Type *base)

Check if RX buffer watermark is exceeded.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – The RX buffer watermark has been exceeded.
- false – The RX buffer watermark has not been exceeded.

static inline uint32_t XSPI_GetRxBufferAvailableBytesCount(XSPI_Type *base)

Get RX buffer available bytes count.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

The available counts of bytes in RX buffer.

static inline uint32_t XSPI_GetRxBufferRemovedBytesCount(XSPI_Type *base)

Get counts of bytes already removed from RX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.

Returns

Counts of removed bytes.

static inline bool XSPI_CheckIPAccessGranted(XSPI_Type *base)

Check if IP access is granted by XSPI arbitration.

Deprecated:

Use *XSPI_CheckIpRequestGranted()* as instead

Parameters

- base – XSPI peripheral base address.

Return values

- true – The IP access is granted arbitration.
- false – No IP access is queued.

static inline bool XSPI_CheckIpWriteTriggered(XSPI_Type *base)

Check if IP write access is triggered.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – The IP write access is granted arbitration.
- false – No IP write access is queued.

static inline bool XSPI_CheckIpReadTriggered(XSPI_Type *base)

Check if IP read access is triggered.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – The IP read access is granted arbitration.
- false – No IP read access is queued.

static inline bool XSPI_CheckFSMValid(XSPI_Type *base)

Check if IPS transfer is granted arbitration or execution.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – Valid, the IPS transfer is granted.
- false – Not valid, no IPS transfer is queued.

static inline bool XSPI_CheckTxBuffLockOpen(XSPI_Type *base)

Check if IP manager can write to TX buffer.

Parameters

- base – **[in]** XSPI peripheral base address.

Return values

- true – TX buffer lock is open.
- false – TX buffer lock is not open.

static inline void XSPI_EnableInterrupts(XSPI_Type *base, uint64_t mask)

Enables the XSPI interrupts.

Parameters

- base – XSPI peripheral base address.
- mask – XSPI interrupt source mask.

static inline void XSPI_DisableInterrupts(XSPI_Type *base, uint64_t mask)

Disables the XSPI interrupts.

Parameters

- base – XSPI peripheral base address.
- mask – XSPI interrupt source mask.

static inline void XSPI_SetSFPFradEALMode(XSPI_Type *base, *xspi_exclusive_access_lock_mode_t* ealMode, uint8_t fradId)

Set exclusive access lock mode for the specific FRAD.

Parameters

- base – **[in]** XSPI peripheral base address.
- ealMode – **[in]** Specify the exclusive access lock mode.
- fradId – **[in]** Specify the FRAD.

static inline *status_t* XSPI_WriteBlocking(XSPI_Type *base, uint8_t *buffer, size_t size)

Sends a buffer of data bytes using blocking method.

Parameters

- base – XSPI peripheral base address.
- buffer – The data bytes to send.
- size – The number of data bytes to send.

Returns
Status of the operation.

static inline *status_t* XSPI_ReadBlocking(XSPI_Type *base, uint8_t *buffer, size_t size)

Receives a buffer of data bytes using a blocking method.

Parameters

- base – XSPI peripheral base address.
- buffer – The data bytes to receive.
- size – The number of data bytes to receive.

Returns
Status of the operation.

void XSPI_Cache64_SetRegionConfig(CACHE64_POLSEL_Type *base, *xspi_cache64_region_config_t* *config)

Set policy of cache regions.

Parameters

- base – CACHE64_POLSEL peripheral base address.
- config – Pointer to the structure *xspi_cache64_region_config_t*.

struct __xspi_device_status_reg_info

#include <fsl_xspi.h> The structure of external device's status register information(location, priority).

Note: Only useful when AHB write to flash device is enabled, and use hardware to clear ppw flag.

Public Members

uint8_t wipLocation

WIP bit field's location.

bool upperHalfWordSelected

Specify which half word(upper or lower) is selected.

bool value1Expired

Specify the value to be check o selected wip location to determine if the page progame wait period is expired, true means expected value is 1, false means expected value is 0.

union __xspi_dll_para

#include <fsl_xspi.h> The union of user custommed DLL parameters, in case of *kXSPI_BypassMode* bypassModePara is used, in case of *kXSPI_AutoUpdateMode* autoUpdateModoPara is used.

Public Members

struct *_xspi_dll_para* bypassModePara

struct *_xspi_dll_para* autoUpdateModoPara

struct __xspi__dll__config

#include <fsl_xspi.h> The structure of Delay-lock-loop configuration.

Public Members

xspi_dll_mode_t dllMode

Dll mode auto update mode or bypass mode

bool useRefValue

Specify to use reference value or not, if the reference code do not satisfy the specific usecase, it is application's responsibility to set custom parameters

- **true:** following inputs are ignored, use the reference values provided by NXP as instead;
- **false:** following inputs take effect.

xspi_dll_para_t dllCustomPara

User custommed parameter to configure DLL, different DLL mode corresponding to different parameters, please refer to the union *xspi_dll_para_t*.

uint8_t dllCustomDelayTapNum

User custommed number of delay elements for each delay tap.

bool enableCdl8

Enable course delay line 8.

struct __xspi__data__learning__config

#include <fsl_xspi.h> The configuration of data learning.

Note: Data learning feature only supported in the DQS sampling method(kXSPI_SampleClkFromExternalDQS).

Public Members

uint32_t pattern

Pre-defined pattern to match.

bool deviceSupported

Specify if external device support data learning feature.

xspi_data_learning_pad_select_t padSelected

Used to select pad which use for pattern matching IO.

struct __xspi_sample_clk_config

#include <fsl_xspi.h> Sample clock configuration, which used to sample data from external device.

Public Members

xspi_sample_clk_source_t sampleClkSource

Specify the sample clock source.

bool enableDQSLatency

Enable DQS latency or not.

xspi_dll_config_t dllConfig

Specify the DLL configuration, to improve data accuracy, please adjust DLL settings based on specific use.

union __xspi_device_interface_settings

#include <fsl_xspi.h> The union of external device's settings, if external device following hyper bus protocol *hyperBusSettings* take effects, if external device following SPI protocol *strandardExtendedSPISettings* take effects.

Public Members

struct *_xspi_device_interface_settings* strandardExtendedSPISettings

struct *_xspi_device_interface_settings* hyperBusSettings

struct __xspi_device_ddr_config

#include <fsl_xspi.h> The structure of device DDR configuration.

Public Members

bool enableDdr

Used to enable/disable DDR mode.

xspi_ddr_data_aligned_clk_t ddrDataAlignedClk

Specify the DDR data aligned clock.

bool enableByteSwapInOctalMode

Swap byte order in octal mode.

struct __xspi_device_config

#include <fsl_xspi.h> External device configuration items.

Public Members

uint32_t xspiRootClk
: XSPI serial root clock.

bool enableCknPad
: Enable/disable clock on differential CKN pad for flash memory A.

xspi_device_interface_type_t deviceInterface
: Type of external device's interface: hyperBus or Strandard/Extended SPI.

xspi_device_interface_settings_t interfaceSettings
: Settings of specified interface.

uint8_t CSHoldTime
: CS line hold time.

uint8_t CSSetupTime
: CS line setup time.

xspi_sample_clk_config_t sampleClkConfig
: Configuration of sample clock.

xspi_device_ddr_config_t *ptrDeviceDdrConfig
: Set as NULL to set device as SDR mode, to change to DDR mode, this member should be populated.

xspi_device_addr_mode_t addrMode
: Address mode of external device.

uint8_t columnAddrWidth
: Width of column address.

bool enableCASInterleaving
: Usually enabled in dual-die device.

uint32_t deviceSize[XSPI_SFAD_COUNT2]
: Size of external device, the unit is KB.

xspi_device_status_reg_info_t *ptrDeviceRegInfo
: Pointer to the structure to store external device's WIP register information, should set as NULL if AHB page write is not used or if AHB page write is used but use software policy to clear flag.

struct _xspi_mdad_config
: *#include <fsl_xspi.h>* MDAD configuration.

Public Members

bool assignIsValid

Specify if the MDAD configuration for the target group is valid

bool enableDescriptorLock

This field provides a means to make the MDAD descriptor read-only. Once written '1' this field will remain '1' until hard reset

uint8_t maskType

0b - ANDed mask 1b - ORed mask

uint8_t mask

Defines the 6-bit mask value for the ID-Match comparison

uint8_t masterIdReference

Specifies the reference value of the Master-ID for MID-comparison

xspi_secure_attribute_t secureAttribute

Defines the secure attribute selection criteria for entry into descriptor queue

struct __xspi__sfp__mdad__config

#include <fsl_xspi.h> The structure of SFP MDAD configurations for all target groups.

Public Members

xspi_mdad_config_t tgMdad[(2U)]

Specify MDAD configurations for each target group.

struct __xspi__frad__tg__config

#include <fsl_xspi.h> FRAD configuration.

Public Members

uint8_t tgMasterAccess[(2U)]

This field define the access restrictions for respective Master Domain corresponding to this FRAD region. Access permissions are decided based on secure an privilege attributes of current transaction. Read access is not restricted.

bool assignIsValid

This field indicates whether the FRAD Descriptor for a specific flash region is valid.

xspi_descriptor_lock_t descriptorLock

This field enables masking of accidental write on FRAD registers. Lock is enabled/disabled by Secure/ Privileged master.

xspi_exclusive_access_lock_mode_t exclusiveAccessLock

This field provides exclusive write lock over a FRAD region based on MDnACP.

struct _xspi_frad_config_t

#include <fsl_xspi.h> FRAD configuration.

Public Members

uint32_t startAddress

Specifies the specific flash memory region starting address

uint32_t endAddress

Specifies the specific flash memory region end address

struct _xspi_sfp_frad_config

#include <fsl_xspi.h> The structure of SFP FRAD configurations.

Public Members

xspi_frad_config_t fradConfig[(8U)]

Specify FRAD configuration for each item.

struct _xspi_frad_transaction_info

#include <fsl_xspi.h> The structure of latest FRAD transaction information.

Public Members

uint32_t startAddr

Latest IP access start address.

uint8_t masterId

Latest IP access master Id.

bool isSecureAccess

Latest IP access is secure access or not.

bool isPrivilegeAccess

Latest IP access is privilege access or not.

bool isCompError

Indicates the error status of the flash region specific comparision check for the latest transaction.

bool isCompValid

Indicates availability of the result or status of a flash region-specific comparison check.

struct __xspi__ip__access__config

#include <fsl_xspi.h> Configurations of IP access(including IP read and write).

Public Members

xspi_sfp_mdad_config_t *ptrSfpMdadConfig

Pointer to the variable in type of *xspi_sfp_mdad_config_t* to set SFP MDAD feature, set as NULL to disable SFP MDAD feature.

xspi_sfp_frad_config_t *ptrSfpFradConfig

Pointer to the variable in type of *xspi_sfp_frad_config_t* to set SFP FRAD feature, set as NULL to disable SFP FRAD

uint32_t sfpArbitrationLockTimeoutValue

Specify the time in IPS clock cycles before an unreleased arbitration triggers a timeout error.

uint32_t ipAccessTimeoutValue

Specify the maximum time in IPS clock cycles for XSPI to wait fro an ongoing write or read command to complete before terminating the command.

struct __xspi__tg__add__write__status

#include <fsl_xspi.h> The status of latest target group address write operation.

Public Members

uint8_t managerId

Manager Id of TG address write operation.

bool secureWrite

TG address write operation is secure write.

bool privilegedWrite

TG address write operation is privileged.

struct __xspi__transfer

#include <fsl_xspi.h> Transfer structure used for XSPI functional interface.

Public Members

uint32_t deviceAddress

Operation device address.

xspi_command_type_t cmdType

Execution command type.

uint8_t seqIndex
Sequence ID for command.

uint32_t *data
Data buffer.

size_t dataSize
Data size in bytes.

xspi_target_group_t targetGroup
Target group.

struct __xspi__handle
#include <fsl_xspi.h> Transfer handle structure for XSPI.

Public Members

uint32_t state
Internal state for XSPI transfer

xspi_target_group_t tgId
Target Group.

uint8_t *data
Data buffer.

size_t dataSize
Remaining Data size in bytes.

size_t transferTotalSize
Total Data size in bytes.

xspi_transfer_callback_t completionCallback
Callback for users while transfer finish or error occurred

void *userData
XSPI callback function parameter.

size_t curTransferSize
Transfer size of current IP access.

struct __xspi__ahbBuffer__sub__buffer__config
#include <fsl_xspi.h> The structure of sub-buffer configurations.

Public Members

xspi_subbuffer_division_factor_t divFactor

Specify the divide factor for current sub-buffer.

uint32_t startAddr

Specify the start address of current sub-buffer.

uint32_t endAddr

Specify the end address of current sub-buffer.

bool enableAhbMonitor

Enable/disable AHB performance monitor for current sub-buffer.

struct _xspi_ahbBuffer_config

#include <fsl_xspi.h> Structure of AHB buffer configurations.

Note: When an AHB read access comes it is assigned to a buffer based on its master ID. Then it further assigned to a sub-buffer based on which sub-buffer address range this transaction lies into.

Note: When sub-buffer division is hit, in case of a buffer miss the controller will fetch the data equal to the size of that sub-buffer.

Note: If sub-buffer division is enabled for a buffer the hit/miss check is done at sub-buffer level. And if prefetch is enable, the prefetch takes place for individual sub-buffers.

Public Members

uint8_t masterId

Specify the ID of the AHB master to be associated with buffer.

xspi_ahbBuffer_sub_buffer_config_t *ptrSubBuffer0Config

Pointer to sub buffer0's configuration.

xspi_ahbBuffer_sub_buffer_config_t *ptrSubBuffer1Config

Pointer to sub buffer1's configuration.

xspi_ahbBuffer_sub_buffer_config_t *ptrSubBuffer2Config

Pointer to sub buffer2's configuration.

xspi_ahbBuffer_sub_buffer_config_t *ptrSubBuffer3Config

Pointer to sub buffer3's configuration.

struct _xspi_ahbBuffer_perf_monitor_result

#include <fsl_xspi.h> The result of AHB buffer performance monitor, including buffer miss count and buffer hit count.

Public Members

uint16_t bufferMissCount

Count of AHB read buffer miss events.

uint16_t bufferHitCount

Count of AHB read buffer hit events.

struct _xspi_ahb_write_config

#include <fsl_xspi.h> The configuration of AHB write access.

Public Members

bool blockSequenceWrite

Enable/disable sequence write operation after page program start.

bool blockRead

Enable/disable read operation after page program start, can be false for RWW extern device.

xspi_ppw_flag_clear_policy_t policy

Specify the policy to clear ppw flag, hardware or software.

uint32_t pageWaitTimeoutValue

Specify, in AHB clock cycles, how long XSPI waits before triggering a read of the device status register, only useful in hardare policy.

uint8_t AWRSeqIndex

Specify the sequence index for AHB write.

uint8_t ARDSRSeqIndex

Specify the sequence index for AHB read status, only useful if either sequence write or read is blocked.

struct _xspi_ahb_access_config

#include <fsl_xspi.h> The structure of AHB access configurations, including AHB buffer settings, AHB Read Seq ID, AHB write settings.

Public Members

xspi_ahbBuffer_config_t buffer[4]

AHB buffer size.

uint8_t ARDSeqIndex

Specify the sequence index for AHB read data.

bool enableAHBPrefetch

Enable/disable AHB read prefetch feature, when enabled, XSPI will fetch more data than current AHB burst.

bool enableAHBBufferWriteFlush

Set true to enable flush of AHB buffer, due to either AHB write or IP access to avoid stable data in AHB buffer.

xspi_ahb_split_size_t ahbSplitSize

Specify ahb split size, set as kXSPI_AhbSplitSizeDisabled if do not want to use this feature.

xspi_ahb_alignment_t ahbAlignment

AHB access towards flash is broken if this AHB alignment boundary is crossed, only support XSPI0 XSPI1

xspi_ahb_write_config_t *ptrAhbWriteConfig

Set as NULL if AHB write feature is not used.

bool enableWriteTerminate

True to enable an AHB transaction can terminate the ongoing AHB read-prefetch, in default it is enabled.

struct __xspi__ahb__read__error__info

#include <fsl_xspi.h> The structure of information when AHB read error occur.

Public Members

uint32_t errorAddr

AHB read error address.

uint8_t errMasterId

AHB read error master Id.

struct __xspi__ahb__request__suspend__info

#include <fsl_xspi.h> The structure of information when AHB reqest is suspended.

Public Members

xspi_ahb_request_suspend_state_t state

Current AHB request state.

uint8_t subBufferId

Suspended AHB request's sub buffer Id.

uint8_t ahbBufferId

Suspended AHB request's buffer Id.

uint16_t dataLeft

How many data left to transfer.

uint32_t address

Current address of suspended AHB request.

struct _xspi_config

#include <fsl_xspi.h> XSPI configuration structure.

Public Members

xspi_byte_order_t byteOrder

Byte ordering endianness

bool enableDoze

Enable/disable doze mode support.

xspi_ahb_access_config_t *ptrAhbAccessConfig

Pointer to AHB access configuration, can be NULL is AHB access is not used.

xspi_ip_access_config_t *ptrIpAccessConfig

Pointer to IP access configuration, can be NULL is IP access is not used.

struct _xspi_cache64_config

#include <fsl_xspi.h> The structure of cache64 configuration.

Region0 range from base address to region0TopAddr[28:10] + 0x3FF, region1 range from region0TopAddr[28:10] + 0x400 to region1TopAddr[28:10] + 0x3FF, region1TopAddr[28:10] + 0x400 to end address.

Public Members

uint32_t region0TopAddr

Top address of region0, bit0-bit9 ignored.

xspi_cache64_policy_t region0Policy

Policy of region0.

uint32_t region1TopAddr

Top address of region1, bit0-bit9 ignored.

xspi_cache64_policy_t region1Policy

Policy of region1.

xspi_cache64_policy_t region2Policy

Policy of region2.

struct bypassModePara

Public Members

uint16_t delayElementCoarseValue
: Delay element coarse adjustment value, range from 0 to 15.

uint16_t delayElementFineValue
: Delay element fine adjustment value, range from 0 to 7.

uint16_t offsetDelayElementCount
: Specify the number of offset delay elements, range from 0 to 15.

uint16_t enableHighFreq
: If set as true, high frequency used for delay chain operations, otherwise low frequency is used.

uint16_t bypassModeReserved
: Reserved.

struct autoUpdateModoPara

Public Members

uint16_t referenceCounter
: Specify count of interval for DLL phase detection, range from 0 to 15.

uint16_t resolution
: Specify minimum resolution for the DLL phase detctor, range from 0 to 15.

uint16_t offsetDelayElementCount
: Specify the number of offset delay elements, range from 0 to 15.

uint16_t tDiv16OffsetDelayElementCount
: Specify the number of T/16 offset elements in the incoming DQS, range from 0 to 15.

uint16_t enableHighFreq
: If set as true, high frequency used for delay chain operations, otherwise low frequency is used.

struct strandardExtendedSPISettings

Public Members

uint32_t pageSize
: The size of page to program, the unit is byte.

struct hyperBusSettings

Public Members

xspi_hyper_bus_x16_mode_t x16Mode
Specify hyper bus X16 mode.

bool enableVariableLatency
If enabled, the count of latency is depends on hyper bus device.

bool forceBit10To1
Force bit 10 to logic one or not.

uint32_t pageSize
The size of page to program, the unit is byte.

union enaPri

Public Members

bool enablePriority
High Priority Enable, it can be written '1' only when OTFAD is disabled, the AHB prefetch feature must be enabled if priority mechanism is enabled.

bool enableAllMaster
When set, buffer3 acts as an all-master buffer.buff[i] routed to buffer3

struct ahbErrorPayload

Public Members

uint32_t highPayload
High 32bit error payload.

uint32_t lowPayload
Low 32bit error payload.

2.93 XSPI eDMA Driver

enum __xspi__edma__ntransfer__size
eDMA transfer configuration
Values:

enumerator kXSPI__EDMAnSize1Bytes
Source/Destination data transfer size is 1 byte every time

enumerator kXSPI_EDMAnSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kXSPI_EDMAnSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kXSPI_EDMAnSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kXSPI_EDMAnSize32Bytes

Source/Destination data transfer size is 32 bytes every time

typedef struct *_xspi_edma_handle* xspi_edma_handle_t

typedef void (*xspi_edma_callback_t)(XSPI_Type *base, *xspi_edma_handle_t* *handle, *status_t* status, void *userData)

XSPI eDMA transfer callback function for finish and error.

typedef enum *_xspi_edma_ntransfer_size* xspi_edma_transfer_nsize_t

eDMA transfer configuration

FSL_XSPI_EDMA_DRIVER_VERSION

XSPI EDMA driver version 2.0.6.

void XSPI_TransferCreateHandleEDMA(XSPI_Type *base, *xspi_edma_handle_t* *handle, *xspi_edma_callback_t* callback, void *userData, *edma_handle_t* *txDmaHandle, *edma_handle_t* *rxDmaHandle)

Initializes the XSPI handle for transfer which is used in transactional functions and set the callback.

Parameters

- base – XSPI peripheral base address
- handle – Pointer to xspi_edma_handle_t structure
- callback – XSPI callback, NULL means no callback.
- userData – User callback function data.
- txDmaHandle – User requested DMA handle for TX DMA transfer.
- rxDmaHandle – User requested DMA handle for RX DMA transfer.

void XSPI_TransferUpdateSizeEDMA(XSPI_Type *base, *xspi_edma_handle_t* *handle, *xspi_edma_transfer_nsize_t* nsize)

Update XSPI EDMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).

See also:

xspi_edma_transfer_nsize_t .

Parameters

- base – XSPI peripheral base address

- handle – Pointer to xspi_edma_handle_t structure
- nsize – XSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXPSI_EDMAnSize1Bytes(one byte).

status_t XSPI_TransferEDMA(XSPI_Type *base, *xspi_edma_handle_t* *handle, *xspi_transfer_t* *xfer)

Transfers XSPI data using an eDMA non-blocking method.

This function writes/receives data to/from the XSPI transmit/receive FIFO. This function is non-blocking.

Parameters

- base – XSPI peripheral base address.
- handle – Pointer to xspi_edma_handle_t structure
- xfer – XSPI transfer structure.

Return values

- kStatus_XSPI_Busy – XSPI is busy transfer.
- kStatus_InvalidArgument – The watermark configuration is invalid, the watermark should be power of 2 to do successfully EDMA transfer.
- kStatus_Success – XSPI successfully start edma transfer.

void XSPI_TransferAbortEDMA(XSPI_Type *base, *xspi_edma_handle_t* *handle)

Aborts the transfer data using eDMA.

This function aborts the transfer data using eDMA.

Parameters

- base – XSPI peripheral base address.
- handle – Pointer to xspi_edma_handle_t structure

status_t XSPI_TransferGetTransferCountEDMA(XSPI_Type *base, *xspi_edma_handle_t* *handle, size_t *count)

Gets the transferred counts of transfer.

Parameters

- base – XSPI peripheral base address.
- handle – Pointer to xspi_edma_handle_t structure.
- count – Bytes transfer.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

edma_handle_t *txDmaHandle

eDMA handler for XSPI Tx.

edma_handle_t *rxDmaHandle

eDMA handler for XSPI Rx.

size_t transferSize

Bytes need to transfer.

xspi_edma_transfer_nsize_t nsize

eDMA SSIZE/DSIZE in each transfer.

uint32_t nbytes

eDMA minor byte transfer count initially configured.

uint8_t count

The transfer data count in a DMA request.

uint32_t state

Internal state for XSPI eDMA transfer.

xspi_edma_callback_t completionCallback

A callback function called after the eDMA transfer is finished.

void *userData

User callback parameter

struct __xspi__edma__handle

#include <fsl_xspi_edma.h> XSPI DMA transfer handle, users should not touch the content of the handle.

Chapter 3

Middleware

3.1 Boot

3.1.1 MCUboot in MCUXpresso SDK

Overview

This repository is a fork of MCUboot for MCUXpresso SDK delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK. This is MCUboot version 2.4.0

MCUboot in MCUXpresso SDK is a fork of the original project that aims to integrate HW features that haven't yet been covered by the upstream code. Currently these are:

- flash remapping techniques for zero-overhead image swapping
- support for encrypted XIP
- use of NXP's SB3 file

Documentation

- Overall details of MCUXpresso SDK can be reviewed here: MCUXpresso SDK Online Documentation
- MCUXpresso SDK examples based on MCUboot are described here: MCUXpresso SDK OTA Examples
- Visit MCUboot - Documentation to review details on the original contents in this sub-repo

Setup

Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution

Contributions are not currently accepted. If the intended contribution is not related to NXP specific code, consider contributing directly to the upstream MCUboot project. Once this MCUboot

fork is synchronized with the upstream project, such contributions will end up here as well. If the intended contribution is a bugfix or improvement for NXP porting layer or for code added or modified by NXP, please open an issue or contact NXP support.

3.2 Connectivity

3.2.1 lwIP

This is the NXP fork of the lwIP networking stack.

- For details about changes and additions made by NXP, see *NXP lwIP Fork Changelog*.
- For details about the NXP porting layer, see *The NXP lwIP Port*.
- For usage and API of lwIP, use official documentation.

MCUXpresso SDK : mcuxsdk-middleware-lwip

Overview This repository is a fork of lwIP TCP/IP stack (http://git.savannah.gnu.org/cgit/lwip.git) for MCUXpresso SDK delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation Overall details can be reviewed here: MCUXpresso SDK Online Documentation

Visit lwIP - Documentation to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution Contributions are not currently accepted. If the intended contribution is not related to NXP specific code, consider contributing directly to the upstream lwIP project. Once this lwIP fork is synchronized with the upstream project, such contributions will end up here as well. If the intended contribution is a bugfix or improvement for NXP porting layer or for code added or modified by NXP, please open an issue or contact NXP support.

NXP Fork This fork of lwIP contains specific modifications and enhancements for NXP MCUXpresso SDK integration, including:

- Adaptation layers for some of the network interfaces found on NXP MCU platforms.
- Adaptation layer supporting compilers used with MCUXpresso SDK.
- Operating system abstraction layers for FreeRTOS and bare-metal.
- NXP implementation of HTTP server.
- Bugfixes and modifications of upstream code, which could not be upstreamed directly.
- CMake and Kconfig files for integration with MCUXpresso SDK build system.

See *changelog* for details.

The NXP lwIP Port

Below is description of possible settings of the port layer and an overview of a few helper functions.

The best place for redefinition of any mentioned macro is lwipopts.h.

The declaration of every mentioned function is in ethernetif.h. Please check the doxygen comments of those functions before.

Link state Physical link state (up/down) and its speed and duplex must be read out from PHY over MDIO bus. Especially link information is useful for lwIP stack so it can for example send DHCP discovery immediately when a link becomes up.

To simplify this port layer offers a function ethernetif_probe_link() which reads those data from PHY and forwards them into lwIP stack.

In almost all examples this function is called every ETH_LINK_POLLING_INTERVAL_MS (1500ms) by a function probe_link_cyclic().

By setting ETH_LINK_POLLING_INTERVAL_MS to 0 polling will be disabled. On FreeRTOS, probe_link_cyclic() will be then called on an interrupt generated by PHY. GPIO port and pin for the interrupt line must be set in the ethernetifConfig struct passed to ethernetif_init(). On bare metal interrupts are not supported right now.

Rx task To improve the reaction time of the app, reception of packets is done in a dedicated task. The rx task stack size can be set by ETH_RX_TASK_STACK_SIZE macro, its priority by ETH_RX_TASK_PRIO.

If you want to save memory you can set reception to be done in an interrupt by setting ETH_DO_RX_IN_SEPARATE_TASK macro to 0.

Disabling Rx interrupt when out of buffers If ETH_DISABLE_RX_INT_WHEN_OUT_OF_BUFFERS is set to 1, then when the port gets out of Rx buffers, Rx enet interrupt will be disabled for a particular controller. Everytime Rx buffer is freed, Rx interrupt will be enabled.

This prevents your app from never getting out of Rx interrupt when the network is flooded with traffic.

ETH_DISABLE_RX_INT_WHEN_OUT_OF_BUFFERS is by default turned on, on FreeRTOS and off on bare metal.

Limit the number of packets read out from the driver at once on bare metal. You may define macro ETH_MAX_RX_PKTS_AT_ONCE to limit the number of received packets read out from the driver at once.

In case of heavy Rx traffic, lowering this number improves the realtime behaviour of an app. Increasing improves Rx throughput.

Setting it to value < 1 or not defining means “no limit”.

Helper functions If your application needs to wait for the link to become up you can use one of the following functions:

- ethernetif_wait_linkup()- Blocks until the link on the passed netif is not up.
- ethernetif_wait_linkup_array() - Blocks until the link on at least one netif from the passed list of netifs becomes up.

If your app needs to wait for the IPv4 address on a particular netif to become different than "ANY" address (255.255.255.255) function ethernetif_wait_ipv4_valid() does this.

NXP lwIP Fork Changelog

This file documents the changes and additions made in the NXP fork of lwIP. For the list of changes in the upstream project version which this fork is based on, see *CHANGELOG*.

Source code included in this SDK is currently based on development version 2.2.2 taken from 3rd party lwIP GIT repository. The webpage https://git.savannah.nongnu.org/cgit/lwip.git allows to browse that repository and also contains URLs for its cloning.

The development versions (X.Y.Z.dev) do not refer to a single source code snapshots. To avoid ambiguity, change log below contains SHA-1 hashes of GIT commits used when importing the upstream code into the SDK.

KSDK refers to Kinetis SDK, the predecessor of MCUXpresso SDK.

2.2.2_rev2 (newest) New features:

- Added support for LWIP_ENET_QOS_FLEXIBLE_CONFIGURATION in enet_ethernetif_qos to allow changing 'enet_qos_config_t' struct.

2.2.2_rev1 New features:

- Synced to upstream lwIP master (2026-06-16, SHA-1: 3d896ba0a37ff3ce73270ca5e230707fe47f60e3) to MCUXpresso SDK.

Bug fixes:

- snmpv3: fixed handling of packets with invalid msgAuthenticationParameters length in snmp_parse_inbound_frame (stack-based buffer overflow / DoS; BDSA, MCUX-89594 / upstream commit 0c957ec0).
- tcp_in: added bounds check for next TCP option byte.
- nd6: parse all IPv6 ND options instead of only the first.
- mdns: fixed copying of invalid search result domains.
- smtp: fixed server-driven AUTH line overflow into tx_buf.
- ip4_frag/ip6_frag: fixed potential NULL-pointer access on memory errors.

2.2.1_rev12 Bug fixes:

- Fixed uninitialized chg_report_tmr in mld6_get_or_alloc_data, this could cause the MLDv6 report to be sent after a long time.

2.2.1_rev11 Bug fixes:

- altcp_tls (mbedTLS): Fixed three related defects in altcp_mbedtls_handle_rx_appldata() that affected reception of large TLS records.

  - Permanent TCP receive window stall when a single TLS record is larger than the inner TCP receive window (TCP_WND) and therefore spans more than one TCP segment: on MBEDTLS_ERR_SSL_WANT_READ the handler now pre-acknowledges the encrypted bytes already consumed by bio_recv, so the inner TCP window reopens and the remaining segments of the record can arrive.
  - LWIP_ASSERT("bogus byte counts", ...) could fire on the record-completion path once the pre-ack above was in effect, because bio_bytes_read is no longer guaranteed to be greater than bio_bytes_appl; the assertion was replaced with a guarded subtraction that only acks the residual TLS overhead when bio_bytes_read > bio_bytes_appl.
  - Truncated/deferred application data when the decrypted plaintext of a single record exceeded PBUF_POOL_BUFSIZE (typically with default MBEDTLS_SSL_IN_CONTENT_LEN of 16 KB): the receive loop now keeps decrypting while mbedtls_ssl_read() returns plaintext, instead of exiting as soon as state->rx/state->rx_app are empty. The loop is bounded by mbedTLS's internal record buffer and the availability of encrypted input on state->rx.

- altcp_tls (mbedTLS): Removed per-connection auto-free of the client altcp_tls_config in altcp_mbedtls_dealloc(). The TLS config lifetime is owned by the application, which calls altcp_tls_free_config() once no remaining connections reference it. The previous behavior caused a use-after-free / double-free when a single client config was shared across multiple connections.

2.2.1_rev10 Bug fixes:

- Checking the value of USE_RTOS macro in porting layer, not only if the macro is defined.

New features:

- The function clock_gettime() renamed to lwiperf_clock_gettime() to avoid conflict with system libs in compilation.

2.2.1_rev9 New features:

- The feature "Iperf now tries to place buffers into fast memory" (introduced in 2.2.1_rev5) can be disabled by defining LWIPERF_QUICKACCESS_TX_BUFFER to 0.

2.2.1_rev8 New features:

- Applied patch #10465: Overall altcp_tls_mbedtls fixes and enhancements (https://savannah.nongnu.org/patch/?10465):
  - Copy received TCP flags
  - Free client's TLS config on dealloc
  - Fix abort management. Fix to return ERR_ABRT on altcp_abort(). Add check to altcp_mbedtls_abort().
  - Free SSL cache and session tickets on tls_free_config()
  - Mark freed members to prevent double free.
  - Correctly close and free LISTEN connections
  - Use handshake steps rather than a single blocking call
  - Add mbedtls return values.

  - free rx_app on dealloc
  - correct ALTCP_MBEDTLS_PLATFORM_ALLOC configuration
  - Manage write apiflags
  - Port to mbedtls v3
  - Fix the return value of altcp_mbdtls_sndbuf when the underlying tcp sndbuf is consumed by ssl expansion bytes.
- Switched from mbedTLS 2.x to mbedTLS 3.x:
  - Updated Kconfig dependencies.
  - Updated altcp_tls_mbedtls layer to use PSA crypto random number generator abstraction instead of ctr_drbg, which would be initialized twice.
  - Updated NXP web server (src/apps/httpsrv).

Bug fixes:

- Fixed sys_sem_new in FreeRTOS porting layer:
  - If the initial count argument was 1, the newly created semaphore was not signalled from the start.
  - The initial count should be just 0 or 1 according to lwIP documentation, but the implementation allowed it to be more than 1 (counting semaphore). Fixed to allow only binary semaphore behavior.
  - src/apps/httpsrv updated to use FreeRTOS API directly where it cannot use sys_sem_t as a counting semaphore anymore.

2.2.1_rev7 New features:

- Operating system abstraction layer: the default implementation of the sys_msleep function (using a temporary allocated semaphore) is overwritten by a new implementation in the adaptation layer (using vTaskDelay on FreeRTOS).

Bug fixes:

- ENET QoS adaptation layer: initialize the CSR (control and status register) clock field in the driver configuration. This fixes the bug with the number of clock ticks for one-microsecond reference timer being set incorrectly.
- Ethernet adaptation layers: Updated logic in ethernetif_pbuf_free_safe to allow the thread to sleep between attempts to schedule pbuf_free on tcpip_thread. Previously, when ETH_RX_TASK_PRIO was higher than tcpip_thread priority and scheduling failed due to resource constraints, ethernetif_pbuf_free_safe would loop indefinitely, preventing tcpip_thread from freeing those resources. Changes include:
  - Added thread sleep between scheduling attempts to prevent infinite loops
  - Removed looping on schedule failure in interrupt context (now asserts instead)
  - Added direct pbuf_free calls when LWIP_ALLOW_MEM_FREE_FROM_OTHER_CONTEXT is enabled

2.2.1_rev6 New features:

- Support for on-demand timers for some of the protocols. Reused from Espressif open source lwIP port. LwIP default protocol timer handlers are invoked periodically, even if there is nothing to do. This is not suitable for low-power applications entering into a sleep mode. The added feature calculates the time when the timer callback should be invoked, instead of waking-up periodically to only check if there is anything to do or not. The feature is turned off by default and can be enabled by the following options:
  - LWIP_IP4_REASSEMBLY_TIMERS_ONDEMAND
  - LWIP_IP6_REASSEMBLY_TIMERS_ONDEMAND
  - LWIP_DNS_TIMERS_ONDEMAND
  - LWIP_DHCP_FINE_TIMERS_ONDEMAND
  - LWIP_IGMP_TIMERS_ONDEMAND
  - LWIP_MLD6_TIMERS_ONDEMAND

  Note that other protocols don't have on-demand timers implemented, so for example if there are open TCP connections, its protocol timers will be still invoked periodically.
- NETC adaptation layer: Possibility to use switch port, see NETC_USE_SWT and related options.
- New option ETH_USE_GPIO_ADAPTER which can be used on platforms without GPIO adapter to bypass the code using it.
- New option ETH_ENET_QOS_MII_MODE to override default MII mode for ENET QoS.

2.2.1_rev5 New features:

- Iperf now tries to place buffers into fast memory.

2.2.1_rev4 New features:

- Enabled build of ALTCP_TLS:
  - This enables option to use ALTCP with mbedTLS for transparent TLS.
  - Added new MCUX_COMPONENT_middleware.lwip.apps.altcp_tls component to Kconfig.
  - Updated CMake to compile the source files if the component is enabled.
  - Added ALTCP_MBEDTLS configuration settings to Kconfig.
- Added support for NETC VSI pre-init.
- Removed unused files with LGPL license or unknown origin.

Bug fixes:

- src/apps/altcp_tls: fixed compilation when MBEDTLS_HAVE_TIME is not defined.
- src/core/ipv6/mld6.c: fixed the setting of MLDv6 general report delay. Especially when the maximum response delay was smaller than MLD6_TMR_INTERVAL, it could be assigned to a value out of the requested range, or end up not sending the report at all.
- Fixed ethernet transmission issue on USB (not handling chained pbufs).

2.2.1_rev3 New features:

- Added optional ability to generate lwIP options header file from Kconfig. The feature has been introduced to align with the changes in MCUXpresso SDK. It can be enabled when using Kconfig and lwIP is used directly from repository, for example when using west build command to generate or build projects. It cannot be used when working with SDK packages, but since lwIP examples (in lwip_examples folder) have been switched to this approach in repository, you can find pregenerated lwipopts_gen.h files in SDK packages (note that if you want to add/change some of the pregenerated settings, do it in lwipopts_gen.h, not lwipopts.h).

  The optional feature is disabled by default and can be enabled by setting CONFIG_MCUX_PRJSEG_middleware.lwip.generated_config=y in prj.conf. When enabled, lwIP options are configured using the Kconfig system, usually written in prj.conf file(s) or .config file with CONFIG_ prefix. Kconfig system uses this information to generate lwipopts_gen.h during build phase. It is included from lwipopts.h (from template/kconfig). The lwipopts.h file is not intended to be edited, it does preprocessing of the values from lwipopts_gen.h.

  The list of options, its types, default values and description is in the file named Kconfig, under the menu "Options (lwipopts_gen.h)".

  Note that there are some differences from using manually edited lwipopts.h:

  – Use CONFIG_ prefix in Kconfig, so for example CONFIG_LWIP_IPV6 instead of LWIP_IPV6. In the generated header, the prefix is removed.

  – For boolean options, use y or n instead of 0 and 1 in Kconfig. For example CONFIG_LWIP_IPV6=y instead of #define LWIP_IPV6 1.

  – Default values for some integer options are using formulas referencing other options. For example, TCP_WND default is (4 * TCP_MSS). Kconfig does not support this, so this option has string type instead of int. The consequence is that you must write quotes in Kconfig for such settings, even if entering integer value, for example CONFIG_TCP_WND="5840". There are more such options, please pay attention to the type definition in Kconfig and any Kconfig warnings in the output of processing.

  – Default values for some options are referencing other options, without using formulas. For example TCP_TTL defaults to IP_DEFAULT_TTL. But since Kconfig translates defaults to values, if you change IP_DEFAULT_TTL later, you may need to override the TCP_TTL to the same value as well (if you still want it the same as IP_DEFAULT_TTL).

  – Options which are function macros, taking no parameters, have a single space between the braces. For example the following written in manually edited lwipopts.h:

    #define LWIP_ASSERT_CORE_LOCKED() sys_check_core_locking()

    would be like this in Kconfig instead:

    CONFIG_LWIP_ASSERT_CORE_LOCKED( )="sys_check_core_locking()"

  – Some functions macros are used in the lwIP code if they are defined. Because Kconfig generates string options with empty value, it would result in a compilation error. Therefore, such options have _CUSTOM suffix in Kconfig and the final option is hidden, without a prompt, generated only when the one with the _CUSTOM suffix does not have empty string value. Example is LOCK_TCPIP_CORE option, where it could be written as CONFIG_LOCK_TCPIP_CORE_CUSTOM( )="sys_lock_tcpip_core()" instead of CONFIG_LOCK_TCPIP_CORE( )="sys_lock_tcpip_core()".

  – Function signatures cannot be written in the generated lwipopts_gen.h. But it can be written in CONFIG_LWIP_OPT_H_CODE_LINE_N options, where N is a line number. It will appear in template/kconfig/lwipopts.h before lwipopts_gen.h is included. That

way the signatures will be visible for the code which uses the function macros. For example the options to bring some function signatures from NXP porting layer could be entered like this for FreeRTOS projects:

```
CONFIG_LWIP_OPT_H_CODE_LINE_1="void sys_lock_tcpip_core(void);"
CONFIG_LWIP_OPT_H_CODE_LINE_2="void sys_unlock_tcpip_core(void);"
CONFIG_LWIP_OPT_H_CODE_LINE_3="void sys_check_core_locking(void);"
CONFIG_LWIP_OPT_H_CODE_LINE_4="void sys_mark_tcpip_thread(void);"
```

or for bare-metal projects:

```
CONFIG_LWIP_OPT_H_CODE_LINE_1="void sys_check_core_locking(void);"
```

Note that the "line numbers" must be used from 1 and numbers cannot be skipped. So it is not possible to comment or delete CONFIG_LWIP_OPT_H_CODE_LINE_2 and continue with CONFIG_LWIP_OPT_H_CODE_LINE_3, all must be renumbered. The function signatures must use only basic C data types, no data types or typedefs from stdint.h or lwIP etc. There might be a cast used in the function macro option if needed.

- Some boolean options, which are just defined without value in C code, must still be written with a value. Example is LWIP_DEBUG, it would be written as CONFIG_LWIP_DEBUG=y or CONFIG_LWIP_DEBUG=n. Other such options are LWIP_PROVIDE_ERRNO and LWIP_ENET_FLEXIBLE_CONFIGURATION.
- Note that Kconfig does not generate boolean values if they are set to "n". The file template/kconfig/lwipopts.h defines the non-generated options to 0, so that lwIP does not define them again to 1 (if it has such default value in its code).
- When using Kconfig, do not define LWIP_RAND. It's function macro is defined in the porting layer in cc.h. See LWIP_DEFINE_LWIP_RAND_IN_CC_H there and in Kconfig.
- All options from opt.h and some of the applications and porting layer options have been enabled Kconfig. If you find something missing, such macro can be added to a project command line or manually edited in lwipopts_gen.h (if it is used and will not be regenerated).
- Formatting macros like X8_F, U16_F, etc. are not present in Kconfig system and does not need to be put into lwipopts.h, see below.
- Default values for formatting macros like X8_F, U16_F, etc. are defined in cc.h, so it does not need to be entered in lwipopts.h.

Bug fixes:

- src/apps/mdns: mdns_resp_remove_netif stops all timeouts related to the netif now. This prevents the timeout handlers to send answers on netifs which already have mDNS responder disabled and prevents access to deallocated structures. It also prevents excessive number of timeouts being scheduled when mDNS responder is disabled and reenabled on netif quickly.

2.2.1_rev2 Bug fixes:

- src/apps/lwiperf: Fixed calculation of the number of UDP datagrams to transmit. When the desired rate was high and datagram size small, the ideal delay between each transmitted datagram could be smaller than 1 microsecond. The value was truncated to zero and resulted in zero-division and/or no UDP transmitted at all.
- port/arch/cc.h: code which checks for the presence of Newlib standard C library adds import of a standard header first so the __NEWLIB__ macro is always available when Newlib is

used. This prevents LWIP_TIMEVAL_PRIVATE from having different values in one program.

2.2.1_rev1 New features:

- Ported lwIP 2.2.1.dev (2024-02-19, branch: master, SHA-1: d0efd9ef7ba08e54b46b1060e2b4629a4907391b) to MCUXpresso SDK.
- Added ETH_MAX_RX_PKTS_AT_ONCE macro. See *port/README.md* for details.
- In port/netc_ethernetif.c, added NETC_VSI_NUM_USED macro to support using VSI. A thread of SI message handling will be started to handle VSI-PSI messages.

Bug fixes:

- Added the missing implementation for IP_FORWARD_ALLOW_TX_ON_RX_NETIF option in the function ip6_forward. Therefore, IPv6 packets could be sent back out on the netif where they were originally received from.
- NETC adaptation layer: Do not call xEventGroupSetBits from ISR.
- Ethernet adaptation layers: Default value of priority of the receive task (ETH_RX_TASK_PRIO) is set lower than the priority of the FreeRTOS daemon task (timer task).

2.2.0_rev11 New features:

- NETC adaptation layer: Possible to disable IPv4/TCP/UDP checksum validation done in HW.
- EtherCAT EoE(Ethernet over EtherCAT) driver is added to lwip.

Bug fixes:

- src/apps/httpsrv/httpsrv_supp.c: Fixed performing of the HTTP server task priority limitation.

2.2.0_rev10 New features:

- Ported lwIP 2.2.0 (2023-09-25, branch: master, SHA-1: 0a0452b2c39bdd91e252aef045c115f88f6ca773, tag: STABLE-2_2_0_RELEASE) to MCUXpresso SDK.
- Enabled hardware-accelerated CRC computation and verification (MAC, IPv4, TCP, UDP, ICMPv4, ICMPv6) for ENET Kinetis, ENET QoS and ENET LPC.
- Enabled link state detection based on PHY interrupts. The ETH_LINK_POLLING_INTERVAL_MS macro controls this - setting it to 0 and specifying ethernetif_config_t->phyIntGpio enables it, setting it to a value greater than zero enables polling instead. Supported only under an RTOS (NO_SYS == 0). By default, the link state is polled.
- ND6: Implemented RFC 4191 type C host, which means default router list (learned from Router Advertisement messages) has been replaced with routing table, which contains default route records for each router and also routes learned from received Route Information Options. Changes partially based on patch 10114. The option LWIP_ND6_NUM_ROUTERS has been removed, and the new option LWIP_ND6_NUM_ROUTES has been added to configure the size of the routing table.

- IPv6: Implemented a new hook - LWIP_HOOK_IP6_CANFORWARD. This hook can be used, for example, for multicast forwarding between netifs. Defining this hook enables multicast traffic forwarding, thus the hook is also invoked for multicast traffic.
- MLD6: Multicast Listener Discovery v1 replaced by v2 (RFC 3810) but without support of source specific multicast.
- port/enet_ethernetif_kinetis.c: Added check to generate/validate ICMPv6 checksum in SW as the Kinetis ENET peripheral does not do it.
- Added disabling of Rx interrupt when the port is out of Rx buffers. See *port/README.md* for more details.

Bug fixes:

- src/apps/lwiperf: Fixed access to invalid data when UDP report is to be sent from a timer but abort has been called before.
- src/apps/lwiperf: Fixed deallocation of TCP server started by client (in reverse or dual modes) which failed to connect.
- port/netc_ethernetif.c: Fixed cache control enablement macro (FSL_SDK_ENABLE_DRIVER_CACHE_CONTROL > FSL_ETH_ENABLE_CACHE_CONTROL).
- port/sys_arch.c: The function sys_assert does not call portENTER_CRITICAL when called from an interrupt.
- src/core/ipv4/ip4.c: Fixed checksum reset condition.
- ND6:
  - Lladdr length is now taken from netif->hwaddr_len so ND6 works properly regardless of NETIF_MAX_HWADDR_LEN.
  - Added check of sufficient length of lladdr options from incoming messages.
- src/apps/httpsrv/httpsrv.c: Fixed hangup in HTTPSRV_release if caller's task has higher priority than server task.
- port/arch/cc.h: LWIP_PLATFORM_DIAG is defined (and can be overridden) independently of the LWIP_DEBUG setting. Removed printing extra newline symbols from LWIP_PLATFORM_DIAG.
- src/apps/lwiperf: The "end of test" UDP datagram is resent more often. This increases the probability of the server to receive it and end the test when datagrams are getting lost.
- Added *port/README.md* describing possible settings and helper functions in the port layer.

2.2.0_rev9 New features:

- Ported lwIP 2.2.0.dev (2023-01-03, branch: master, SHA-1: 3fe8d2fc43a9b69f7ed28c63d44a7744f9c0def9) to MCUXpresso SDK.
- Applied patch to allow sending IPv6 router advertisement. Improved to allow selection of interface and router life time and to allow sending route information options.
- src/apps/lwiperf: Support for reverse test (client receives, server sends). Requires iperf version 2.1.0 or newer.

Bug fixes:

- src/apps/httpsrv: Fixed operation with LWIP_IPV6 enabled. Server can be also accessed using both IPv4 and IPv6 at the same time if compiled with both LWIP_IPV4=1 and LWIP_IPV6=1. Note the type of the field struct httpsrv_param_struct.address has changed from struct sockaddr to struct sockaddr_storage.

2.2.0_rev8 New features:

- src/apps/lwiperf: Added new parameter "buffer_len" to functions lwiperf_start_tcp_client() and lwiperf_start_udp_client() to configure TCP/UDP packet size.
- src/apps/lwiperf: Added new parameter "tos" to functions lwiperf_start_tcp_client() to configure TCP packet priority.
- NETC adaptation layer: Not forcing the RX/TX buffers placement in non-cacheable memory. Requires the symbol FSL_ETH_ENABLE_CACHE_CONTROL to be defined on project level if the memory region, where the buffers are placed by a linker, has cache enabled.

Bug fixes:

- src/apps/httpsrv: Added missing includes.
- src/apps/lwiperf: Fixed TCP client to send settings at the beginning of each 128 KB block like the PC iperf 2.0.x application does.
- src/apps/lwiperf: Fixed validation of TCP received data (with LWIPERF_CHECK_RX_DATA enabled, works with iperf 2.0.x).
- src/apps/lwiperf: Fixed lwiperf_list_remove() to clear references to the removed item.
- src/apps/lwiperf: Program does not assert when buffer cannot be cloned in UDP test, only "can't clone buffer" message is printed.

2.2.0_rev7 New features:

- Ported lwIP 2.2.0.dev (2022-05-09, branch: master, SHA-1: 239918ccc173cb2c2a62f41a40fd893f57faf1d6) to MCUXpresso SDK.
- Added function ethernetif_probe_link() which reads actual link, speed and duplex settings from phy and passes them to driver. Stack could be set to call this function periodically by setting ETH_LINK_POLLING_INTERVAL_MS to value higher than zero.
- Added helper functions ethernetif_wait_linkup() and ethernetif_wait_ipv4_valid() to allow blocking of RTOS task or bare metal application until link is up or IPv4 address becomes valid.
- Added NETC adaptation layer.
- Processing of rx packets under RTOS moved from ISR to a separate task to improve system reaction times. Switch back to old behavior can be done by setting ETH_DO_RX_IN_SEPARATE_TASK macro to 0.

Bug fixes:

- port: Fixed copying of pbuf contents. Previous code was using an incorrect end condition and could result in the overrun of the destination buffer if more packets were on the queue.
- port: Delegating pbuf_free calls to tcpip_thread via pbuf_free_callback where possible (RTOS), ensuring pbuf_free is not called from interrupt context when LWIP_ALLOW_MEM_FREE_FROM_OTHER_CONTEXT is not set (bare metal).
- port/enet_ethernetif_qos.c - Fixed ENET_RXBD_NUM which was used instead of ENET_TXBD_NUM.
- port/enet_ethernetif_qos.c - Fixed buffer alignment to be at least 64.
- src/apps/lwiperf: Fixed IPv6 TCP TX throughput lower than IPv4 by modifying maximum segment size to avoid sending two segments instead of one.
- src/apps/lwiperf: Out-of-order datagrams in UDP RX server mode are counted to the throughput.

- src/apps/httpsrv: Implemented receive timeouts on sockets.
- src/apps/httpsrv: Don't assert on HTTP session task creation failure.
- src/apps/httpsrv: Fixed build with IPv6 enabled.
- src/apps/httpsrv: Updated endianess macros required for websocket SHA generation.
- src/apps/httpsrv: Added missing includes.

2.2.0_rev6 New features:

- Ported lwIP 2.2.0.dev (2022-03-25, branch: master, SHA-1: 124dc0a64ef5d7c14a27e3115e5888df6559cb72) to MCUXpresso SDK.
- Implemented leaving of multicast groups on ENET and ENET QOS.

2.2.0_rev5 New features:

- Ported lwIP 2.2.0.dev (2021-05-11, branch: master, SHA-1: 7ec4e9be304e7f8953740f10b2c810a292e89449) to MCUXpresso SDK.
- LPC ENET adaptation layer allocates more buffers for frame reception now. Previously the number of receive buffers was determined by ENET_RXBD_NUM, which defaults to 5. It is determined by ENET_RXBUFF_NUM now, which is 2 * ENET_RXBD_NUM by default. Increase was needed because the actual version of LPC ENET driver always hold ENET_RXBD_NUM number of buffers and few additional buffers are needed for passing zero-copy frame data to lwIP. If this takes too much memory in your application, you can counteract by decreasing PBUF_POOL_SIZE, since PBUF_POOL is used only for transmission when LPC ENET, Kinetis ENET or ENET QOS is used.

2.2.0_rev4 New features:

- Ported lwIP 2.2.0.dev (2021-03-05, branch: master, SHA-1: 0056522cc974d2be2005c324f37187b5b3695765) to KSDK 2.0.0.
- LWIP_DHCP_DOES_ACD_CHECK option default changed to 0 (disabled):
  - Although the ACD check makes getting IP address from DHCP more robust, it added several seconds delay at startup of all applications which use DHCP.
  - This feature was not present in earlier versions of lwIP.
- ENET QOS adaptation layer - implemented zero-copy on receive.
- Kinetis ENET and ENET QOS adaptation layers allocate more buffers for frame reception now. Previously the number of receive buffers was determined by ENET_RXBD_NUM, which defaults to 5. It is determined by ENET_RXBUFF_NUM now, which is 2 * ENET_RXBD_NUM by default. Increase was needed because the actual version of Kinetis ENET and ENET QOS drivers always hold ENET_RXBD_NUM number of buffers and few additional buffers are needed for passing zero-copy frame data to lwIP. If this takes too much memory in your application, you can counteract by decreasing PBUF_POOL_SIZE, since PBUF_POOL is used only for transmission when Kinetis ENET or ENET QOS is used.
- Removed ethernetif_config_t.non_dma_memory field which was required to configure memory ranges unusable by ENET DMA on LPC devices. The setting has been replaced by BOARD_ENET_NON_DMA_MEMORY_ARRAY macro.

2.2.0_rev3 New features:

- Ported lwIP 2.2.0.dev (2020-07-07, branch: master, SHA-1: c385f31076b27efb8ee37f00cb5568783a58f299) to KSDK 2.0.0.

2.2.0_rev2 New features:

- Kinetis ENET adaptation layer - implemented zero-copy on receive.
- lwiperf - counter of transferred bytes extended from 32 to 64 bit

Bug fixes:

- Fixed restarting Auto IP from DHCP.

2.2.0_rev1 New features:

- Ported lwIP 2.2.0.dev (2019-12-12, branch: master, SHA-1: 555812dcec38c9a2ef1ef9b31816291549fbf9f8) to KSDK 2.0.0.
- Implemented LWIP_ASSERT_CORE_LOCKED related functions in sys_arch.c. It can be enabled in lwipopts.h:

```
#define LWIP_ASSERT_CORE_LOCKED() sys_check_core_locking()
#define LWIP_MARK_TCPIP_THREAD() sys_mark_tcpip_thread() // if NO_SYS == 0
#define LOCK_TCPIP_CORE() sys_lock_tcpip_core() // if NO_SYS == 0 and LWIP_TCPIP_
↪CORE_LOCKING == 1
#define UNLOCK_TCPIP_CORE() sys_unlock_tcpip_core() // if NO_SYS == 0 and LWIP_
↪TCPIP_CORE_LOCKING == 1
```

2.1.2_rev5 New features:

- Implemented TCP_USER_TIMEOUT socket option.
- Implemented SIOCOUTQ ioctl.

2.1.2_rev4 New features:

- Ported lwIP 2.1.3.dev (2019-02-27, branch: STABLE-2_1_x, SHA-1: 1bb6e7f52de1cd86be0eed31e348431edc2cd01e) to KSDK 2.0.0.
- Updated sys_thread_new implementation and comment.
- Kinetis ENET adaptation layer - reading frames into a pbuf chain is conditionally compiled only when a single pbuf from pool cannot hold maximum frame size (PBUF_POOL_BUFSIZE >= maximum frame size). Avoiding this code also reduces stack size requirements by about 1.5 kilobytes.

Bug fixes:

- Fixes in ethernetif_linkoutput() in enet_ethernetif_lpc.c:
  - Removed access to possibly freed pbuf.
  - Call pbuf_free() when transmit buffers not available.
  - When copying pbuf chain, updating the number of necessary transmit buffers to wait for, which can be often smaller in the copy.

- When CGI script is reading POST data by chunks, the loop in httpsrv_read() may cause blocking in receive function waiting for more data at the end of the stream
  - HTTPSRV_cgi_read() - added limiting of the last chunk length according to content length to avoid undesired blocking
- Applied AUTOIP patch 9847 - with modification to support multiple network interfaces.
- Fixed buffer overflow in httpsrv when application provided CGI script does not handle the whole content of POST request
  - Removed LwipMibCompiler contrib application as it contained LGPL licensed files in SharpSnmpLib.

2.1.2_rev3 New features:

- lwiperf updated with UDP client/server support from the patch 9751

2.1.2_rev2 Bug fixes:

- Fixed lwiperf_abort() in lwiperf.c to correctly close connections and free resources

2.1.2_rev1 New features:

- Ported lwIP 2.1.2 (2018-11-22, SHA-1: 159e31b689577dbf69cf0683bbaffbd71fa5ee10) to KSDK 2.0.0.
- Ported lwIP-contrib 2.1.0 (2018-09-24, SHA-1: 35b011d4cf4c4b480f8859c456587a884ec9d287) to KSDK 2.0.0.

2.0.3_rev1 New features:

- Ported lwIP 2.0.3 (2017-09-15, SHA-1: 92f23d6ca0971a32f2085b9480e738d34174417b) to KSDK 2.0.0.

2.0.2_rev1 New features:

- Ported lwIP 2.0.2 (2017-03-13, SHA-1: c0862d60746e2d1ceae69af4c6f24e469570ecef) to KSDK 2.0.0.

2.0.0_rev3 New features:

- Ported lwIP 2.0.0 (2016-11-10, SHA-1: 216bf89491815029aa15463a18744afa04df58fe) to KSDK 2.0.0.

2.0.0_rev2 New features:

- Ported lwIP 2.0.0 RC2 (2016-08-08, SHA-1: b1dfd00f9233d124514a36a8c8606990016f2ad4) to KSDK 2.0.0.

2.0.0_rev1 New features:

- Ported lwIP 2.0.0 RC0 (2016-05-26) to KSDK 2.0.0.
- Changed lwIP bare-metal examples to use poll-driven approach instead of interrupt-driven one.

1.4.1_rev2 New features:

- Enabled critical sections in lwIP.

Bug fixes:

- Fixed default lwIP packet-buffer size to be able to accept a maximum size frame from the ENET driver.
- Fixed possible drop of multi-frame packets during transmission.

1.4.1_rev1 New features:

- Ported lwIP 1.4.1 to KSDK 2.0.0.

3.3 File System

3.3.1 FatFs

MCUXpresso SDK : mcuxsdk-middleware-fatfs

Overview This repository is for FatFs middleware delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation Overall details can be reviewed here: MCUXpresso SDK Online Documentation

Visit FatFs - Documentation to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution Contributions are not currently accepted. Guidelines to contribute will be posted in the future.

Repo Specific Content This is MCUXpresso SDK fork of FatFs (FAT file system created by ChaN). Official documentation is available at http://elm-chan.org/fsw/ff/

MCUXpresso version is extending original content by following hardware specific porting layers:

- mmc_disk
- nand_disk

- ram_disk
- sd_disk
- sdspi_disk
- usb_disk

Changelog FatFs

All notable changes to this project will be documented in this file.
The format is based on Keep a Changelog

[R0.16_rev0]

- Upgraded to version 0.16
- Applied patch https://elm-chan.org/fsw/ff/patch/ff16p1.diff
- Applied patch https://elm-chan.org/fsw/ff/patch/ffunicode.zip
- Make lb and hb variables conditional. Fix 'unused variable' issue:
  - ffunicode.c:10622:28: error: unused variable 'lb'
  - ffunicode.c:10622:7: error: unused variable 'hb'

[R0.15_rev0]

- Upgraded to version 0.15
- Applied patches from http://elm-chan.org/fsw/ff/patches.html

[R0.14b_rev1]

- Applied patches from http://elm-chan.org/fsw/ff/patches.html

[R0.14b_rev0]

- Upgraded to version 0.14b

[R0.14a_rev0]

- Upgraded to version 0.14a
- Applied patch ff14a_p1.diff and ff14a_p2.diff

[R0.14_rev0]

- Upgraded to version 0.14
- Applied patch ff14_p1.diff and ff14_p2.diff

[R0.13c_rev0]

- Upgraded to version 0.13c
- Applied patches ff_13c_p1.diff,ff_13c_p2.diff, ff_13c_p3.diff and ff_13c_p4.diff.

[R0.13b_rev0]

- Upgraded to version 0.13b

[R0.13a_rev0]

- Upgraded to version 0.13a. Added patch ff_13a_p1.diff.

[R0.12c_rev1]

- Add NAND disk support.

[R0.12c_rev0]

- Upgraded to version 0.12c and applied patches ff_12c_p1.diff and ff_12c_p2.diff.

[R0.12b_rev0]

- Upgraded to version 0.12b.

[R0.11a]

- Added glue functions for low-level drivers (SDHC, SDSPI, RAM, MMC). Modified diskio.c.
- Added RTOS wrappers to make FatFs thread safe. Modified syscall.c.
- Renamed ffconf.h to ffconf_template.h. Each application should contain its own ffconf.h.
- Included ffconf.h into diskio.c to enable the selection of physical disk from ffconf.h by macro definition.
- Conditional compilation of physical disk interfaces in diskio.c.

3.4 Trusted Firmware-M

3.4.1 Trusted-Firmware-M

MCUX SDK Documents

TF-M Documents MCUXpresso SDK : mcuxsdk-middleware-trusted-firmware-m

Overview

This repository is for MCUXpresso SDK trusted-firmware-m middleware delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation

Overall details can be reviewed here: MCUXpresso SDK Online Documentation.

Setup

Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution

Contributions are not currently accepted. Guidelines to contribute will be posted in the future.

Psa-arch-tests Documents MCUXpresso SDK : mcuxsdk-middleware-Platform-Security-Architecture-Test-Suite

This repository is a fork of ARM Platform Security Architecture Test suits(https://github.com/ARM-software/psa-arch-tests)(SHA: 8d8630fdeb2). Modifications have been made to adapt to NXP MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests for the complete delivery of MCUXpresso SDK.

Introduction

This test suite is one of a set of resources provided by Arm that can help organizations develop products that meet the security requirements of PSA Certified on Arm-based platforms. The PSA Certified scheme provides a framework and methodology that helps silicon manufacturers, system software providers and OEMs to develop more secure products. Arm resources that support PSA Certified range from threat models, standard architectures that simplify development and increase portability, and open-source partnerships that provide ready-to-use software. You can read more about PSA Certified here at psacertified web and find more Arm resources here at arm platform security resources.

For more information, visit the PSA webpage

PSA Certified APIs specifications

API specifications are placed in PSA API github site.

PSA Certified API Compliance

The *PSA Certified API tests* are the basis for validating compliance with PSA Certified APIs. For more information on the certification program, see psacertified.org

For more information about passing PSA Certified API compliance, what it means, what it requires, and how it relates to the PSA Certified initiative, please read the *Frequently Asked Questions about PSA Certified API compliance*.

GitHub branch

- For API certification, use the release branch and pick the appropriate release tag.
- To get the latest version of the code with bug fixes and new features, use the master branch.

Architecture test suite

The current implementation of the Architecture test suite contains tests for following PSA Certified specifications. Arm licensees may contact their partner manager to obtain a copy of this specification.

Future versions of the test suite will be enhanced to include tests for other specifications of the Platform Security Architecture.

The tests are available as open source. The tests and the corresponding abstraction layers are available with an Apache v2 license allowing for external contribution.

TBSA-v8M

The test suite for this specification is located in the tbsa-v8m directory of this repository. See *TBSA-v8m Readme* file for more details.

PSA APIs

The test suite for this specification is located in the api-tests directory of this repository. See *PSA APIs Tests Readme* file for more details.

Secure Debug

The test suite for this specification is located in the secure-debug directory of this repository. See *Secure Debug Readme* file for more details.

License

Arm PSA test suite is distributed under Apache v2.0 License.

Feedback, contributions, and support

- For feedback, use the GitHub Issue Tracker that is associated with this repository.
- For support, send an email to support-psa-arch-tests@arm.com with details.
- Arm licensees can contact Arm directly through their partner managers.
- Arm welcomes code contributions through GitHub pull requests.

Copyright (c) 2018-2022, Arm Limited and Contributors. All rights reserved.

Third Party Documents

TF-M Documents Trusted Firmware-M Documentation

Introduction

Trusted Firmware M

Trusted Firmware-M (TF-M) implements the Secure Processing Environment (SPE) for Armv8-M, Armv8.1-M architectures (e.g. the Cortex-M33, Cortex-M23, Cortex-M55, Cortex-M85 processors) and dual-core platforms. It is the platform security architecture reference implementation aligning with PSA Certified guidelines, enabling chips, Real Time Operating Systems and devices to become PSA Certified.

TF-M relies on an isolation boundary between the Non-secure Processing Environment (NSPE) and the Secure Processing Environment (SPE). It can but is not limited to using the Arm TrustZone technology on Armv8-M and Armv8.1-M architectures. In pre-Armv8-M architectures physical core isolation is required.

TF-M consists of:

- Secure Boot to authenticate NSPE and SPE images
- TF-M Core for controlling the isolation, communication and execution within SPE and with NSPE
- Crypto, Internal Trusted Storage (ITS), Protected Storage (PS), Firmware Update and Attestation secure services

TF-M implements PSA-FF-M defined IPC and SFN mechanisms to allow communication between isolated firmware partitions. TF-M is highly configurable allowing users to only include the required secure services and features. Project provides *Base Configuration* build with just TF-M core and platform drivers and 4 predefined configurations known as *TF-M Profiles*. TF-M Profiles or TF-M base can be configured to include required services and features as described in the *Configuration* section.

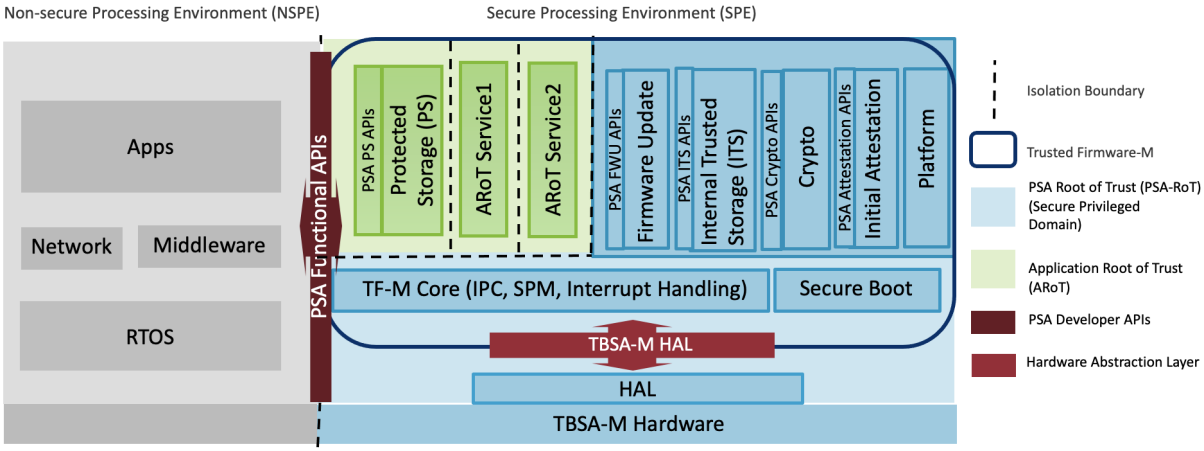


Fig. 1: FF-M compliant design with TF-M

Applications and Libraries in the Non-secure Processing Environment can utilize these secure services with a standardized set of PSA Functional APIs. Applications running on Cortex-M devices can leverage TF-M services to ensure secure connection with edge gateways and IoT cloud services. It also protects the critical security assets such as sensitive data, keys and certificates on the platform. TF-M is supported on several Cortex-M based *Microcontrollers* and Real Time Operating Systems (RTOS).

Terms TFM and TF-M are commonly used in documents and code and both refer to Trusted Firmware M. *Glossary* has the list of terms and abbreviations.

Repositories

TF-M is comprised of multiple repositories that supplement each other in making the project both customisable and maintainable.

Table 1: TF-M Repositories

Repository	Description
trusted-firmware-m	Software implementation of TF-M with documentation and essential tools
tf-m-tests	Tests that focus on the functionalities of TF-M components
tf-m-tools	Non essential tools used for testing and verification of TF-M
tf-m-extras	Extension of the main repository to host examples, demonstrations, third-party modules etc

License

The software is provided under a BSD-3-Clause *License*. Contributions to this project are accepted under the same license with developer sign-off as described in the *Contributing Guidelines*.

This project contains code from other projects as listed below. The code from external projects is limited to bl2, lib and platform folders. The original license text is included in those source files.

- The bl2 folder contains files imported from MCUBoot project and the files have Apache 2.0 license.

- The lib/ext folder may contain 3rd party projects and files with diverse licenses. Here are some that are different from the BSD-3-Clause and may be a part of the runtime image. The source code for these projects is fetched from upstream at build time only.
  - CMSIS_5 - Apache 2.0 license
  - mbedcrypto - Apache 2.0 license MbedTLS
  - mcuboot - Apache 2.0 license MCUBoot
  - qcbor - Modified BSD-3-Clause license
  - tf-m-extras - Set of additional components. Please check individually in tf-m-extras repository
- The platform folder currently contains platforms support imported from the external project and the files may have different licenses.

Supported Platforms

- Cortex-M85 system:
  - Corstone-310 Ethos-U55/U65 Ecosystem FVP.
  - FPGA image loaded on MPS3 board (AN555).
- Cortex-M55 system:
  - FPGA image loaded on MPS3 board (AN547).
  - FPGA image loaded on MPS3 board (AN552).
  - Corstone-300 Ethos-U55/U65 Ecosystem FVP.
- Cortex-M33 system:
  - FPGA image loaded on MPS2 board (AN521).
  - Fast model FVP_MPS2_AEMv8M (AN521).
  - Musca-B1 test chip board.
  - Musca-S1 test chip board.
  - FPGA image loaded on MPS3 board (AN524).
  - LPC55S69.
  - STM32H573I-DK.
  - B-U585I-IOT02A.
  - NUCLEO L552ZE Q.
  - DISCO L562QE.
  - nRF9160 DK (nordic_nrf/nrf9160dk_nrf9160).
  - nRF5340 DK (nordic_nrf/nrf5340dk_nrf5340_cpuapp).
  - BL5340 DVK (lairdconnectivity/bl5340_dvk_cpuapp).
  - MAX32657 (adi/max32657).
- Cortex-M23 system:
  - FPGA image loaded on MPS2 board (AN519).
  - M2351.
  - M2354.
- Dual Core Cortex-M system:
  - PSoc64.

- Runtime Security Engine (RSE):
  - *RSE.*

The document *Platform Deprecation and Removal* lists the deprecated platforms, removed from the upstream.

The document *Supported Platforms* lists the details.

Release Notes and Process

The *Release Cadence and Process* provides release cadence and process information.

The *Releases* provides details of major features of the release and platforms supported.

Feedback and Support

For this release, feedback is requested via email to tf-m@lists.trustedfirmware.org.

A bi-weekly technical forum is available for discussion on any technical topics online. Welcome to join TF-M Forum.

Copyright (c) 2017-2022, Arm Limited. All rights reserved.

Getting Started Guides

First Things First

Prerequisite

Trusted Firmware M provides a reference implementation of platform security architecture reference implementation aligning with PSA Certified guidelines. It is assumed that the reader is familiar with specifications can be found at Platform Security Architecture Resources.

The current TF-M implementation specifically targets TrustZone for ARMv8-M so a good understanding of the v8-M architecture is also necessary. A good place to get started with ARMv8-M is developer.arm.com.

Build and run instructions

Trusted Firmware M source code is available on git.trustedfirmware.org.

To build & run TF-M:

- Follow the this guide to set up and check your environment.
- Follow the *Build instructions* to compile and build the TF-M source.
- Follow the *Run TF-M examples on Arm platforms* for information on running the example.

To port TF-M to a another system or OS, follow the *OS Integration Guide*

Contributing Guidelines contains guidance on how to contribute to this project.

Set up build environments

TF-M officially supports a limited set of build environments and setups. In this context, official support means that the environments listed below are actively used by team members and active developers, hence users should be able to recreate the same configurations by following the instructions described below. In case of problems, the TF-M team provides support only for these environments, but building in other environments can still be possible.

The following environments are supported:

Linux

1. version supported:

   Ubuntu 18.04 x64+

2. install dependencies:

```
sudo apt-get install -y git curl wget build-essential libssl-dev python3 \
python3-pip cmake make
```

3. verify cmake version:

```
cmake --version
```

Note: Please download cmake 3.15 or later version from https://cmake.org/download/.

4. add CMake path into environment:

```
export PATH=<CMake path>/bin:$PATH
```

Windows

1. version supported:

   Windows 10 x64

2. install dependecies:

- Git client latest version (https://git-scm.com/download/win)
- CMake (native Windows version)
- GNU make (http://gnuwin32.sourceforge.net/packages/make.htm)
- Python3 (native Windows version) and the pip package manager (from Python 3.4 it's included)

3. add CMake path into environment:

```
set PATH=<CMake_Path>\bin;%PATH%
```

Install python dependencies

Clone the TF-M source code, and then install the TF-M's additional Python dependencies.

Linux

1. get the TF-M source code:

```
git clone https://git.trustedfirmware.org/TF-M/trusted-firmware-m.git
```

2. TF-M's tools/requirements.txt file declares additional Python dependencies. Install them with pip3:

```
pip3 install --upgrade pip
cd trusted-firmware-m
pip3 install -r tools/requirements.txt
```

Windows

1. get the TF-M source code:

```
git clone https://git.trustedfirmware.org/TF-M/trusted-firmware-m.git
```

2. TF-M's tools/requirements.txt file declares additional Python dependencies. Install them with pip3:

```
cd trusted-firmware-m
pip3 install -r tools\requirements.txt
```

Install a toolchain

To compile TF-M code, at least one of the supported compiler toolchains have to be available in the build environment. The currently supported compiler versions are:

- Arm Compiler v6.13 ~ v6.14, v6.18+

  Linux

  - Download the standalone packages from here.
  - Add Arm Compiler into environment:

    ```
    export PATH=<ARM_CLANG_PATH>/bin:$PATH
    export ARM_PRODUCT_PATH=<ARM_CLANG_PATH>/sw/mappings
    ```

  - Configure proper tool variant and license.

  Windows

  - Download the standalone packages from here.
  - Add Arm Compiler into environment:

    ```
    set PATH=<ARM_CLANG_PATH>\bin;%PATH%
    set ARM_PRODUCT_PATH=<ARM_CLANG_PATH>\sw\mappings
    ```

  - Configure proper tool variant and license.

Note: Arm compiler v6.15 ~ v6.17 may cause MemManage fault. This defect has been fixed since Arm compiler v6.18. See [SDCOMP-59788] in Armclang v6.18 release note for details.

- GNU Arm compiler v7.3.1+

  Linux

  - Download the GNU Arm compiler from here.
  - Add GNU Arm into environment:

    ```
    export PATH=<GNU_ARM_PATH>/bin:$PATH
    ```

  Windows

  - Download the GNU Arm compiler from here.
  - Add GNU Arm into environment:

    ```
    set PATH=<GNU_ARM_PATH>\bin;%PATH%
    ```

Note: GNU Arm compiler version *10-2020-q4-major* has an issue in CMSE support. The bug is reported in here. Select other GNU Arm compiler versions instead.

- IAR Arm compiler v8.42.x, v8.50.x

  Linux

  - Download IAR build tools from here.

- Add IAR Arm compiler into environment:

```
export PATH=<IAR_COMPILER_PATH>/bin:$PATH
```

Windows

- Download IAR build tools from here.
- Add IAR Arm compiler into environment:

```
set PATH=<IAR_COMPILER_PATH>\bin;%PATH%
```

Build AN521 regression sample

Here, we take building TF-M for AN521 platform with regression tests using GCC as an example:

Linux

Get the TF-M tests source code:

```
git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
```

Build SPE and NSPE.

```
cd </tf-m-tests/tests_reg>
cmake -S spe -B build_spe -DTFM_PLATFORM=arm/mps2/an521 -DCONFIG_TFM_SOURCE_PATH=
↪<TF-M source dir absolute path> \
    -DCMAKE_BUILD_TYPE=Debug -DTFM_TOOLCHAIN_FILE=<TF-M source dir absolute path>/
↪toolchain_GNUARM.cmake \
    -DTEST_S=ON -DTEST_NS=ON \
cmake --build build_spe -- install

cmake -S . -B build_test -DCONFIG_SPE_PATH=<tf-m-tests absolute path>/tests_reg/build_spe/api_ns␣
↪\
    -DCMAKE_BUILD_TYPE=Debug -DTFM_TOOLCHAIN_FILE=<tf-m-tests absolute path>/tests_
↪reg/build_spe/api_ns/cmake/toolchain_ns_GNUARM.cmake
cmake --build build_test
```

Windows

Important: Use "/" instead of "\" when assigning Windows paths to CMAKE variables, for example, use "c:/build" instead of "c:\\build".

Get the TF-M tests source code:

```
git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
```

Build SPE and NSPE.

```
cd </tf-m-tests/tests_reg>
cmake -G"Unix Makefiles" -S spe -B build_spe -DTFM_PLATFORM=arm/mps2/an521 -DCONFIG_TFM_
↪SOURCE_PATH=<TF-M source dir absolute path> \
    -DCMAKE_BUILD_TYPE=Debug -DTFM_TOOLCHAIN_FILE=<TF-M source dir absolute path>/
↪toolchain_GNUARM.cmake \
    -DTEST_S=ON -DTEST_NS=ON \
cmake --build build_spe -- install

cmake -G"Unix Makefiles" -S . -B build_test -DCONFIG_SPE_PATH=<tf-m-tests absolute path>/tests_
↪reg/build_spe/api_ns \
    -DCMAKE_BUILD_TYPE=Debug -DTFM_TOOLCHAIN_FILE=<tf-m-tests absolute path>/tests_
↪reg/build_spe/api_ns/cmake/toolchain_ns_GNUARM.cmake
cmake --build build_test
```

Note: The latest Windows support long paths, but if you are less lucky then you can reduce paths by moving the build directory closer to the root by changing the -B option of the commands, for example, to C:\build_spe and C:\build_test folders.

Run AN521 regression sample

Run the sample code on SSE-200 Fast-Model, using FVP_MPS2_AEMv8M provided by Arm Development Studio.

Note: Arm Development Studio is not essential to develop TF-M, you can skip this section if don't want to try on Arm develop boards.

Linux

1. install Arm Development Studio to get the fast-model.

   Download Arm Development Studio from here.

2. Add bl2.axf and tfm_s_ns_signed.bin to symbol files in Debug Configuration menu.

```
<DS_PATH>/sw/models/bin/FVP_MPS2_AEMv8M  \
--parameter fvp_mps2.platform_type=2 \
--parameter cpu0.baseline=0 \
--parameter cpu0.INITVTOR_S=0x10000000 \
--parameter cpu0.semihosting-enable=0 \
--parameter fvp_mps2.DISABLE_GATING=0 \
--parameter fvp_mps2.telnetterminal0.start_telnet=1 \
--parameter fvp_mps2.telnetterminal1.start_telnet=0 \
--parameter fvp_mps2.telnetterminal2.start_telnet=0 \
--parameter fvp_mps2.telnetterminal0.quiet=0 \
--parameter fvp_mps2.telnetterminal1.quiet=1 \
--parameter fvp_mps2.telnetterminal2.quiet=1 \
--application cpu0=<build_spe>/api_ns/bin/bl2.axf \
--data cpu0=<build_test>/tfm_s_ns_signed.bin@0x10080000
```

Note: The log is output to telnet by default. It can be also redirected to stdout by adding the following parameter.

```
--parameter fvp_mps2.UART0.out_file=/dev/stdout
```

To automatically terminate the fast-model when it finishes running, you can add the following parameters:

```
--parameter fvp_mps2.UART0.shutdown_on_eot=1
```

Windows

1. install Arm Development Studio to get the fast-model.

   Download Arm Development Studio from here.

2. Add bl2.axf and tfm_s_ns_signed.bin to symbol files in Debug Configuration menu.

```
<DS_PATH>\sw\models\bin\FVP_MPS2_AEMv8M  \
--parameter fvp_mps2.platform_type=2 \
--parameter cpu0.baseline=0 \
--parameter cpu0.INITVTOR_S=0x10000000 \
--parameter cpu0.semihosting-enable=0 \
```

(continues on next page)

(continued from previous page)

```
--parameter fvp_mps2.DISABLE_GATING=0 \
--parameter fvp_mps2.telnetterminal0.start_telnet=1 \
--parameter fvp_mps2.telnetterminal1.start_telnet=0 \
--parameter fvp_mps2.telnetterminal2.start_telnet=0 \
--parameter fvp_mps2.telnetterminal0.quiet=0 \
--parameter fvp_mps2.telnetterminal1.quiet=1 \
--parameter fvp_mps2.telnetterminal2.quiet=1 \
--application cpu0=<build_spe>/api_ns/bin/bl2.axf \
--data cpu0=<build_test>/tfm_s_ns_signed.bin@0x10080000
```

Note: To automatically terminate the fast-model when it finishes running, you can add the following parameters:

```
--parameter fvp_mps2.UART0.shutdown_on_eot=1
```

After completing the procedure you should see the following messages on the DAPLink UART (baud 115200 8n1):

```
...
#### Execute test suites for the Secure area ####
Running Test Suite PSA protected storage S interface tests (TFM_S_PS_TEST_1XXX)...
> Executing 'TFM_S_PS_TEST_1001'
Description: 'Set interface'
TEST: TFM_S_PS_TEST_1001 - PASSED!
> Executing 'TFM_S_PS_TEST_1002'
Description: 'Set interface with create flags'
TEST: TFM_S_PS_TEST_1002 - PASSED!
> Executing 'TFM_S_PS_TEST_1003'
Description: 'Set interface with NULL data pointer'
TEST: TFM_S_PS_TEST_1003 - PASSED!
> Executing 'TFM_S_PS_TEST_1005'
Description: 'Set interface with write once UID'
TEST: TFM_S_PS_TEST_1005 - PASSED!
....
```

Tool & Dependency overview

To build the TF-M firmware the following tools are needed:

- C compiler of supported toolchains
- CMake version 3.15 or later
- Git
- gmake, aka GNU Make
- Python v3.x
- a set of python modules listed in tools/requirements.txt

Dependency chain

```
@startuml
 skinparam state {
   BackgroundColor #92AEE0
   FontColor black
   FontSize 16
   AttributeFontColor black
   AttributeFontSize 16
```

(continues on next page)

(continued from previous page)

```
}
state fw as "Firmware" : TF-M binary
state c_comp as "C Compiler" : C99
state python as "Python" : v3.x

fw --> c_comp
fw --> CMake
CMake --> gmake
CMake --> Ninja
fw --> cryptography
fw --> pyasn1
fw --> yaml
fw --> jinja2
fw --> cbor2
fw --> click
fw --> imgtool
c_comp --> GCC
c_comp --> CLANG
c_comp --> IAR
cryptography --> python
pyasn1 --> python
yaml --> python
jinja2 --> python
cbor2 --> python
click --> python
imgtool --> python
kconfiglib --> python
@enduml
```

Next steps

Here are some next steps for exploring TF-M:

- Detailed *Build instructions*.
- *IAR Build instructions*.
- Try other *Samples and Demos*.
- *Documentation generation*.

Copyright (c) 2017-2024, Arm Limited. All rights reserved. Copyright (c) 2022 Cypress Semiconductor Corporation (an Infineon company) or an affiliate of Cypress Semiconductor Corporation. All rights reserved.

Security

Trusted Firmware-M Generic Threat Model

Introduction

This document introduces a generic threat model of Trusted Firmware-M (TF-M). This generic threat model provides an overall analysis of TF-M implementation and identifies general threats and mitigation.

There is also a dedicated document for physical attack mitigations which can be found *here*.

Note: If you think a security vulnerability is found, please follow Trustedfirmware.org [Security-Incident-Process] to contact TF-M security team.

Scope

TF-M supports diverse models and topologies. It also implements multiple isolation levels. Each case may focus on different target of evaluation (TOE) and identify different assets and threats. TF-M implementation consists of several secure services, defined as Root of Trust (RoT) service. Those RoT services belong to diverse RoT (Application RoT or PSA RoT) and access different assets and hardware. Therefore each RoT service may require a dedicated threat model.

The analysis on specific models, topologies or RoT services may be covered in dedicated threat model documents. Those threat models are out of the scope of this document.

Methodology

The threat modeling in this document follows the process listed below to build up the threat model.

- Target of Evaluation (TOE)
- Assets identification
- Data Flow Diagram (DFD)
- Threats Prioritization
- Threats identification

TOE is the entity on which threat modeling is performed. The logic behind this process is to firstly investigate the TOE which could be a system, solution or use case. This first step helps to identify the assets to be protected in TOE.

According to TOE and assets, Trust Boundaries can be determined. The Data Flow Diagram (DFD) across Trust Boundaries is then defined to help identify the threats.

Those threats should be prioritized based on a specific group of principals and metrics. The principals and metrics should also be specified.

Target of Evaluation

A typical TF-M system diagram from a high-level overview is shown below. TF-M is running in the Secure Processing Environment (SPE) and NS software is running in Non-secure Processing Environment (NSPE). For more details, please refer to Platform Security Architecture Firmware Framework for M (FF-M) [FF-M] and FF-M 1.1 Extensions [FF-M-1.1-Extensions].

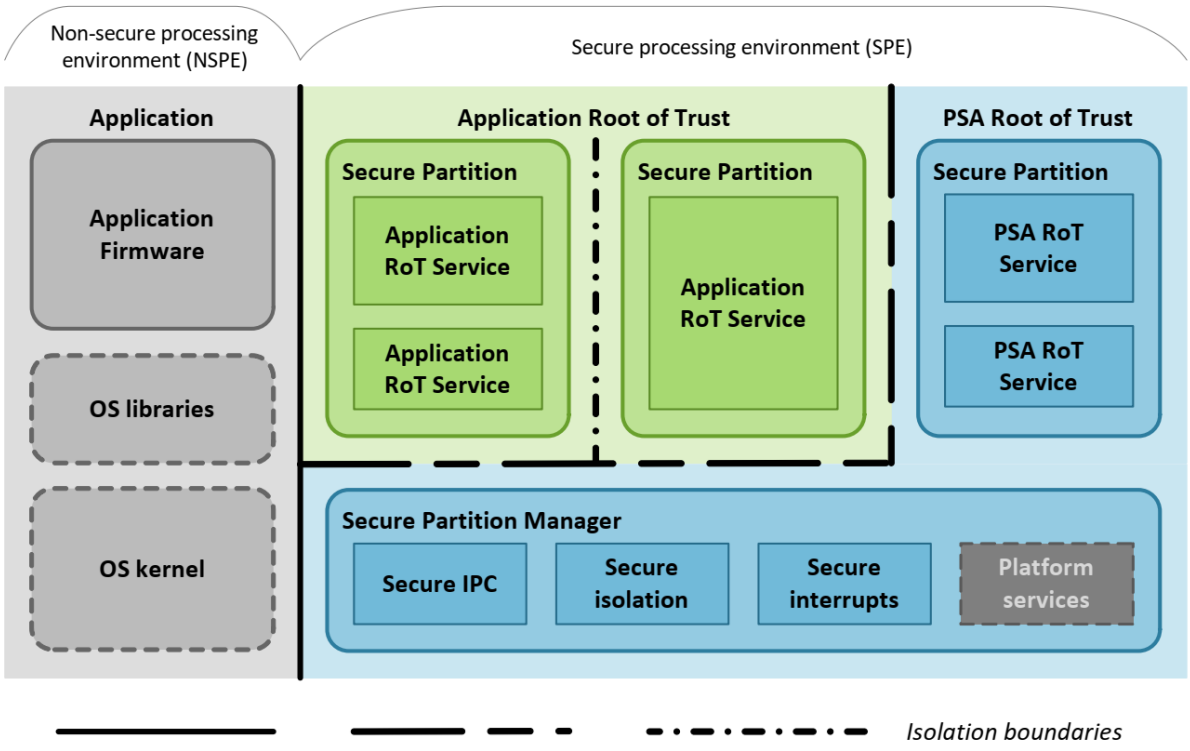
The TOE in this general model is the SPE, including TF-M and other components running in SPE.

The TOE can vary in different TF-M models, RoT services and usage scenarios. Refer to dedicated threat models for the specific TOE definitions.

Asset identification

In this threat model, assets include the general items listed below:

- Hardware Root of Trust data, e.g.
  - Hardware Unique Key (HUK)
  - Root authentication key
  - Other embedded root keys
- Software RoT data, e.g.



  - Secure Partition Manager (SPM) code and data
  - Secure partition code and data
  - NSPE data stored in SPE
  - Data generated in SPE as requested by NSPE
- Availability of entire RoT service
- Secure logs, including event logs

Assets may vary in different use cases and implementations. Additional assets can be defined in an actual usage scenario and a dedicated threat model.

For example, in a network camera use case, the following data can be defined as assets too:

- Certificate for connecting to cloud
- Session keys for encryption/decryption in the communication with cloud
- Keys to encrypt/decrypt the videos and photos

Data Flow Diagram

The Trust Boundary isolates SPE from NSPE, according to the TOE definition in *Target of Evaluation*. The Trust Boundary mapped to block diagram is shown in the figure below. Other modules inside SPE stay in the same TOE as TF-M does.

Valid data flows across the Trust Boundary are also shown in the figure below. This threat model only focuses on the data flows related to TF-M.

More details of data flows are listed below.

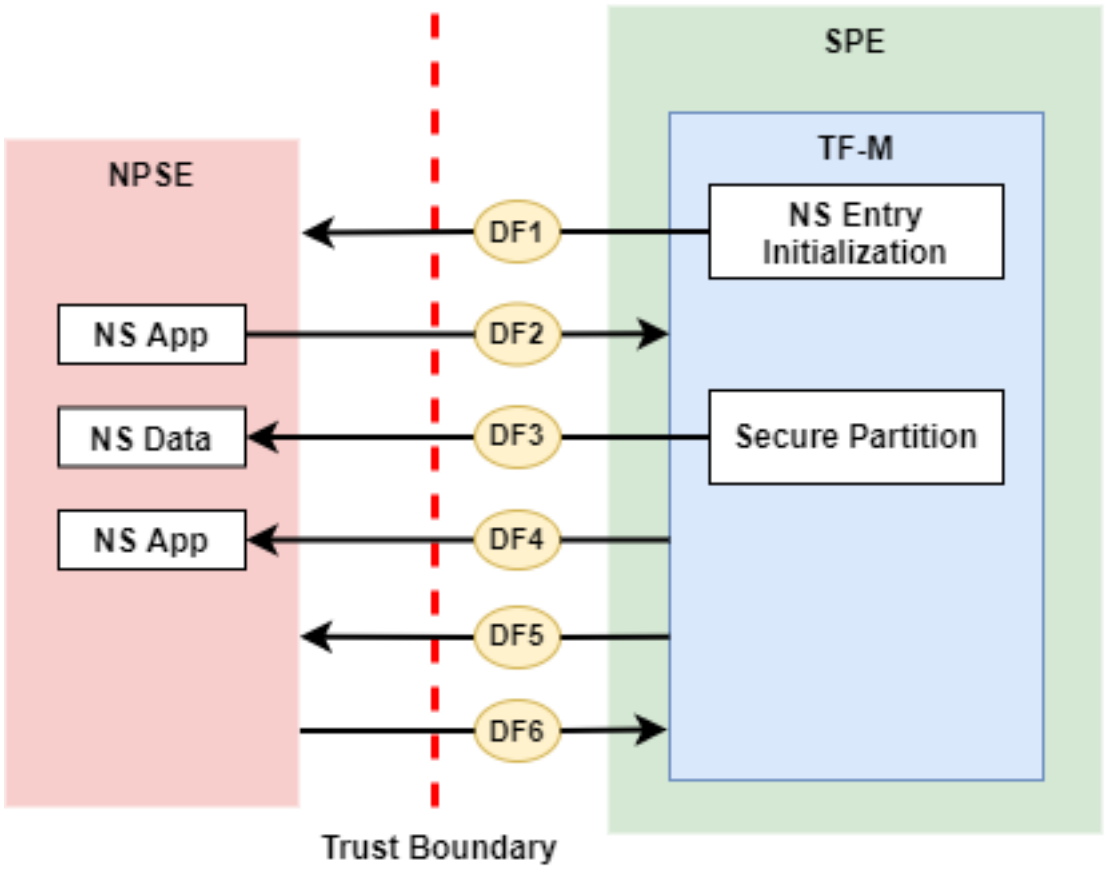


Table 2: TF-M Data Flows between NSPE and SPE

Data flow	Description
DF1	TF-M initializes NS entry and activates NSPE. • On Armv8-M platforms with TrustZone, TF-M will hand over the control to Non-secure state. • On dual-cpu platforms, Secure core starts NS core booting.
DF2	NSPE requests TF-M RoT services. NSPE requests RoT services via PSA Client APIs defined in [FF-M]. In Armv8-M TrustZone scenarios, SG instruction is executed in a Non-secure Callable region to trigger a transition from Non-secure state to Secure state. On dual-cpu platforms, non-secure core sends PSA Client calls to secure core via mailbox.
DF3	Secure Partitions fetch input data from NS and write back output data to NS. As required in [FF-M], Secure Partitions should not directly access NSPE memory. Instead, RoT services relies on TF-M SPM to access NSPE memory.
DF4	TF-M returns RoT service results to NSPE after NS request to RoT service is completed. In Armv8-M TrustZone scenarios, it also triggers a transition from Secure state back to Non-secure state. On dual-cpu platforms, secure core returns the result to non-secure core via mailbox.
DF5	Non-secure interrupts preempt SPE execution in Armv8-M TrustZone scenarios.
DF6	Secure interrupts preempt NSPE execution in

Note: All the other data flows across the Trust Boundary besides the valid ones mentioned above should be prohibited by default. Proper isolation must be configured to prevent NSPE directly accessing SPE.

Threats irrelevant to data flows in *TF-M Data Flows between NSPE and SPE* may be specified in *Miscellaneous threats*.

Data flows inside SPE (informative)

Since all the SPE components stay in the TOE within the same Trust Boundary in this threat model, the data flows between SPE components are not covered in this threat model. Instead, those data flows and corresponding threats will be identified in the dedicated threat model documents of TF-M RoT services and usage scenarios.

Those data flows inside SPE include following examples:

- Data flows between TF-M and BL2
- Data flows between RoT services and SPM
- Data flows between RoT services and corresponding secure hardware and assets, such as secure storage device, crypto hardware accelerator and Hardware Unique Key (HUK).

Threat identification

Threat priority

Threat priority is indicated by the score calculated via Common Vulnerability Scoring System (CVSS) Version 3.1 [CVSS]. The higher the threat scores, the greater severity the threat is with and the higher the priority is.

CVSS scores can be mapped to qualitative severity ratings defined in CVSS 3.1 specification [CVSS_SPEC]. This threat model follows the same mapping between CVSS scores and threat priority rating.

As a generic threat model, this document focuses on *Base Score* which reflects the constant and general severity of a threat according to its intrinsic characteristics.

The *Impacted Component* defined in [CVSS_SPEC] refers to the assets listed in *Asset identification*.

Threats and mitigation list

This section lists generic threats and corresponding mitigation, based on the the analysis of data flows in *Data Flow Diagram*.

Threats are identified following STRIDE model. Please refer to [STRIDE] for more details.

The field CVSS Score reflects the threat priority defined in *Threat priority*. The field CVSS Vector String contains the textual representation of the CVSS metric values used to score the threat. Refer to [CVSS_SPEC] for more details of CVSS vector string.

Note: A generic threat may have different behaviors and therefore require different mitigation, in diverse TF-M models and usage scenarios.

This threat model document focuses on general analysis of the following threats. For the details in a specific configuration and usage scenario, please refer to the dedicated threat model document.

NS entry initialization

This section identifies threats on DF1 defined in *Data Flow Diagram*.

Table 3: TFM-GENERIC-NS-INIT-T-1

Index	**TFM-GENERIC-NS-INIT-T-1**
Description	The NS image can be tampered by an attacker
Justification	An attack may tamper the NS image to inject malicious code
Category	Tampering
Mitigation	By default TF-M relies on MCUBoot to validate NS image. The validation of NS image integrity and authenticity is completed in secure boot before jumping to NS entry or booting up NS core. Refer to [SECURE-BOOT] for more details. The validation may vary in diverse vendor platforms specific Chain of Trust (CoT) implementation.
CVSS Score	3.5 (Low)
CVSS Vector String	CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N

Table 4: TFM-GENERIC-NS-INIT-T-2

Index	**TFM-GENERIC-NS-INIT-T-2**
Description	An attacker may replace the current NS image with an older version.
Justification	The attacker downgrades the NS image with an older version which has been deprecated due to known security issues. The older version image can pass the image signature validation and its vulnerabilities can be exploited by attackers.
Category	Tampering
Mitigation	TF-M relies on MCUBoot to perform anti-rollback protection. TF-M defines a non-volatile counter API to support anti-rollback. Each platform must implement it using specific trusted hardware non-volatile counters. For more details, refer to [ROLLBACK-PROTECT]. The anti-rollback protection implementation can vary on diverse platforms.
CVSS Score	3.5 (Low)
CVSS Vector String	CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N

Table 5: TFM-GENERIC-NS-INIT-T-I-1

Index	**TFM-GENERIC-NS-INIT-T-I-1**
Description	If SPE doesn't complete isolation configuration before NSPE starts, NSPE can access secure regions which it is disallowed to.
Justification	Secure data can be tampered or disclosed if NSPE is activated and accesses secure regions before isolation configuration is completed by SPE.
Category	Tampering/Information disclosure
Mitigation	SPE must complete and enable proper isolation to protect secure regions from being accessed by NSPE, before jumping to NS entry or booting up NS core. TF-M executes isolation configuration at early stage of secure initialization before NS initialization starts. On dual-cpu platform, platform specific initialization must halt NS core until isolation is completed, as defined in [DUAL-CPU-BOOT]. TF-M defines isolation configuration HALs for platform implementation. The specific isolation configuration depends on platform specific implementation.
CVSS Score	9.0 (Critical)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:N

Table 6: TFM-GENERIC-NS-INIT-T-I-2

Index	**TFM-GENERIC-NS-INIT-T-I-2**
Description	If SPE doesn't complete isolation configuration before NSPE starts, NSPE can control devices or peripherals which it is disallowed to.
Justification	On some platforms, devices and peripherals can be configured as Secure state in runtime. If security status configuration of those device and peripherals are not properly completed before NSPE starts, NSPE can control those device and peripherals and may be able to tamper data or access secure data.
Category	Tampering/Information disclosure
Mitigation	SPE must complete and enable proper configuration and isolation to protect critical devices and peripherals from being accessed by NSPE, before jumping to NS entry or booting up NS core. TF-M executes isolation configuration of devices and peripherals at early stage of secure initialization before NS initialization starts. The specific isolation configuration depends on platform specific implementation.
CVSS Score	9.0 (Critical)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:N

Table 7: TFM-GENERIC-NS-INIT-I-2

Index	**TFM-GENERIC-NS-INIT-I-2**
Description	If SPE leaves some SPE information in non-secure memory or shared registers when NSPE starts, NSPE may access those SPE information.
Justification	If NSPE can access those SPE information from shared registers or non-secure memory, secure information may be disclosed.
Category	Information disclosure
Mitigation	SPE must clean up the secure information from shared registers before NS starts. TF-M invalidates registers not banked before handing over the system to NSPE on Armv8-M platforms with TrustZone. On dual-cpu platforms, shared registers are implementation defined, such as Inter-Processor Communication registers. Dual-cpu platforms must not store any data which may disclose secure information in the shared registers. SPE must avoid storing SPE information in non-secure memory.
CVSS Score	4.3 (Medium)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:C/C:L/I:N/A:N

Table 8: TFM-GENERIC-NS-INIT-D-1

Index	**TFM-GENERIC-NS-INIT-D-1**
Description	An attacker may block NS to boot up
Justification	An attacker may block NS to boot up, such as by corrupting NS image, to stop the whole system from performing normal functionalities.
Category	Denial of service
Mitigation	No SPE information will be disclosed and TF-M won't be directly impacted. It relies on NSPE and platform specific implementation to mitigate this threat. It is out of scope of this threat model.
CVSS Score	4.0 (Medium)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L

NSPE requests TF-M secure service

This section identifies threats on DF2 defined in *Data Flow Diagram*.

Table 9: TFM-GENERIC-REQUEST-SERVICE-S-1

Index	**TFM-GENERIC-REQUEST-SERVICE-S-1**
Description	A malicious NS application may pretend as a secure client to access secure data which NSPE must not directly access.
Justification	[FF-M] defines Client ID to distinguish clients which request RoT services. Secure clients are assigned with positive IDs and non-secure clients are assigned with negative ones. A malicious NS application may provide a positive Client ID to pretend as a secure client to access secure data.
Category	Spoofing
Mitigation	TF-M checks the Client ID from NSPE. If the NS Client ID is not a valid one, TF-M will report this as a security error.
CVSS Score	8.4 (High)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N

Table 10: TFM-GENERIC-REQUEST-SERVICE-T-1

Index	**TFM-GENERIC-REQUEST-SERVICE-T-1**
Description	An attacker in NSPE may tamper the service request input or output vectors between check and use (Time-Of-Check-to-Time-Of-Use (TOCTOU)).
Justification	If SPE validates the content in input/output vectors locally in NSPE memory, an attacker in NSPE can have a chance to tamper the content after the validation successfully passes. Then SPE will provide RoT service according to the corrupted parameters and it may cause further security issues.
Category	Tampering
Mitigation	In TF-M implementation, the validation of NS input/output vectors are only executed after those vectors are copied from NSPE into SPE. It prevents an attack from NSPE to tamper those parameters after validation in TF-M.
CVSS Score	7.8 (High)
CVSS Vector String	CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:C/C:H/I:H/A:N

Table 11: TFM-GENERIC-REQUEST-SERVICE-T-2

Index	**TFM-GENERIC-REQUEST-SERVICE-T-2**
Description	A malicious NS application may request to tamper data belonging to SPE.
Justification	A malicious NS application may request SPE RoT services to write malicious value to SPE data. The malicious NS application may try to tamper SPE assets, such as keys, or modify configurations in SPE. The SPE data belongs to components in SPE and must not be accessed by NSPE.
Category	Tampering
Mitigation	TF-M executes memory access check to all the RoT service requests. If a request doesn't have enough permission to access the target memory region, TF-M will refuse this request and assert a security error.
CVSS Score	7.1 (High)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:C/C:N/I:H/A:N

Table 12: TFM-GENERIC-REQUEST-SERVICE-R-1

Index	**TFM-GENERIC-REQUEST-SERVICE-R-1**
Description	A NS application may repudiate that it has requested services from a RoT service.
Justification	A malicious NS application may call a RoT service to access critical data in SPE, which it is disallowed to, via a non-public vulnerability. It may refuse to admit that it has accessed that data.
Category	Repudiation
Mitigation	TF-M implements an event logging secure service to record the critical events, such as the access to critical data.
CVSS Score	0.0 (None)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:C/C:N/I:N/A:N

Table 13: TFM-GENERIC-REQUEST-SERVICE-I-1

Index	**TFM-GENERIC-REQUEST-SERVICE-I-1**
Description	A malicious NS application may request to read data belonging to SPE.
Justification	A malicious NS application may request SPE RoT services to copy SPE data to NS memory. The SPE data belongs to components in SPE and must not be disclosed to NSPE, such as root keys.
Category	Information disclosure
Mitigation	TF-M executes memory access check to all the RoT service requests. If a request doesn't have enough permission to access the target memory region, TF-M will refuse this request and assert a security error.
CVSS Score	7.1 (High)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N

Table 14: TFM-GENERIC-REQUEST-SERVICE-T-I-1

Index	**TFM-GENERIC-REQUEST-SERVICE-T-I-1**
Description	A malicious NS application may request to control secure device and peripherals, on which it doesn't have the permission.
Justification	A malicious NS application may request RoT services to control secure device and peripherals, on which it doesn't have the permission.
Category	Tampering/Information disclose
Mitigation	TF-M performs client check to validate whether the client has the permission to access the secure device and peripherals.
CVSS Score	9.0 (Critical)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:N

Table 15: TFM-GENERIC-REQUEST-SERVICE-D-1

Index	**TFM-GENERIC-REQUEST-SERVICE-D-1**
Description	A Malicious NS applications may frequently call secure services to block secure service requests from other NS applications.
Justification	TF-M runs on IoT devices with constrained resource. Even though multiple outstanding NS PSA Client calls can be supported in system, the number of NS PSA client calls served by TF-M simultaneously are still limited. Therefore, if a malicious NS application or multiple malicious NS applications continue calling TF-M secure services frequently, it may block other NS applications to request secure service from TF-M.
Category	Denial of service
Mitigation	TF-M is unable to manage behavior of NS applications. Assets are not disclosed and TF-M is neither directly impacted in this threat. It relies on NS OS to enhance scheduling policy and prevent a single NS application to occupy entire CPU time. It is beyond the scope of this threat model.
CVSS Score	4.0 (Medium)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L

Table 16: TFM-GENERIC-REQUEST-SERVICE-D-2

Index	**TFM-GENERIC-REQUEST-SERVICE-D-2**
Description	A malicious NS application may provide invalid NS memory addresses as the addresses of input and output data in RoT service requests.
Justification	SPE may be unable to achieve full knowledge of NS memory mapping. SPE may fail to capture those invalid NS memory addresses during memory access check since those invalid addresses may not be included in isolation configuration. In that case, SPE will access those invalid NS memory addresses later to read or write data. It may trigger a system error to crash the whole system immediately. The malicious NS application may be blocked by NS MPU from directly accessing that invalid NS memory address. But it may manipulate SPE to access that address instead.
Category	Denial of service
Mitigation	TF-M executes memory access check to the memory addresses in all the NS requests. On Armv8-M platforms with TrustZone, TF-M invokes TT instructions to execute memory address check. If a NS memory area is not matched in any valid SAU or MPU region, it will be marked as invalid and any access permission is disallowed. Therefore, SPM will reject any NS request containing invalid NS memory addresses and reports it as as a security error. On dual-core platforms, TF-M implements a default memory access check. If a NS memory area is not found in any memory region configured for isolation, it will be marked as invalid and therefore SPM will reject the corresponding NS request. It will be reported as a security error. Dual-core platforms may implement platform specific memory check to replace the default one. It relies on platform specific implementation to capture invalid memory address. It is out of the scope of this document.
CVSS Score	3.2 (Low)
CVSS Vector String	CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:C/C:N/I:N/A:L

RoT services read and write NS data

This section identifies threats on DF3 defined in *Data Flow Diagram*.

RoT services can either directly access NS memory or rely on TF-M SPM to obtain NS input data and send response data back to NS memory.

Table 17: TFM-GENERIC-SECURE-SERVICE-RW-T-1

Index	**TFM-GENERIC-SECURE-SERVICE-RW-T-1**
Description	An attacker may tamper NS input data while the RoT service is processing those data.
Justification	A RoT service may access NS input data multiple times during its data processing. For example, it may validate or authenticate the NS input data before it performs further processing. If the NS input data remains in NSPE memory during the RoT service execution, an attacker may tamper the NS input data in NSPE memory after the validation passes.
Category	Tampering
Mitigation	If RoT services request SPM to read and write NS data. TF-M SPM follows [FF-M] to copy the NS input data into SPE memory region owned by the RoT service, before the RoT service processes the data. Therefore, the NS input data is protected during the RoT service execution from being tampered. If RoT services can directly access NS memory and read NS input data multiple times during data processing, it is required to review and confirm the implementation of the RoT service copies NS input data into SPE memory area before it processes the data.
CVSS Score	3.2 (Low)
CVSS Vector String	CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:C/C:N/I:L/A:N

Table 18: TFM-GENERIC-SECURE-SERVICE-RW-T-2

Index	**TFM-GENERIC-SECURE-SERVICE-RW-T-2**
Description	A malicious NS application may embed secure memory addresses into a structure in RoT service request input vectors, to tamper secure memory which the NS application must not access.
Justification	[FF-M] limits the total number of input/output vectors to 4. If a RoT service requires more input/output vectors, it may define a parameter structure which embeds multiple input/output buffers addresses. However, as a potential security risk, a malicious NS application can put secure memory addresses into a valid parameter structure to bypass TF-M validation on those memory addresses. The parameter structure can pass TF-M memory access check since itself is valid. However, if the RoT service parses the structure and directly write malicious data from NSPE to the secure memory addresses in parameter structure, the secure data will be tampered.
Category	Tampering
Mitigation	It should be avoided to embed memory addresses into a single input/output vector. If more than 4 memory addresses are required in a RoT service request, it is recommended to split this request into two or multiple service calls and therefore each service call requires no more than 4 input/output vectors. If RoT services request SPM to read and write NS data. SPM will validate the target addresses and can detect the invalid addresses to mitigate this threat. If RoT services can directly access NS memory, it is required to review and confirm the implementation of RoT service request doesn't embed memory addresses.
CVSS Score	7.1 (High)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:C/C:N/I:H/A:N

Table 19: TFM-GENERIC-SECURE-SERVICE-RW-I-1

Index	**TFM-GENERIC-SECURE-SERVICE-RW-I-1**
Description	Similar to *TFM-GENERIC-SECURE-SERVICE-RW-T-2*, a malicious NS application can embed secure memory addresses in a parameter structure in RoT service request input vectors, to read secure data which the NS application must not access.
Justification	Similar to the description in *TFM-GENERIC-SECURE-SERVICE-RW-T-2*, the secure memory addresses hidden in the RoT service input/output vector structure may bypass TF-M validation. Without a proper check, the RoT service may copy secure data to NSPE according to the secure memory addresses in structure, secure information can be disclosed.
Category	Information disclosure
Mitigation	It should be avoided to embed memory addresses into a single input/output vector. If more than 4 memory addresses are required in a RoT service request, it is recommended to split this request into two or multiple service calls and therefore each service call requires no more than 4 input/output vectors. If RoT services request SPM to read and write NS data. SPM will validate the target addresses and can detect the invalid addresses to mitigate this threat. If RoT services can directly access NS memory, it is required to review and confirm the implementation of RoT service request doesn't embed memory addresses.
CVSS Score	7.1 (High)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N

TF-M returns secure service result

This section identifies threats on DF4 defined in *Data Flow Diagram*.

When RoT service completes the request from NSPE, TF-M returns the success or failure error code to NS application.

In Armv8-M TrustZone scenarios, TF-M writes the return code value in the general purpose register and returns to Non-secure state.

On dual-cpu platforms, TF-M writes the return code to NS mailbox message queue via mailbox.

Table 20: TFM-GENERIC-RETURN-CODE-I-1

Index	**TFM-GENERIC-RETURN-CODE-I-1**
Description	SPE may leave secure data in the registers not banked after the SPE completes PSA Client calls and executes BXNS to switch Armv8-M back to Non-secure state.
Justification	If SPE doesn't clean up the secure data in registers not banked before switching into NSPE in Armv8-M core, NSPE can read the SPE context from those registers.
Category	Information disclosure
Mitigation	In Armv8-M TrustZone scenarios, TF-M cleans general purpose registers not banked before switching into NSPE to prevent NSPE probing secure context from the registers. When FPU is enabled in TF-M, secure FP context belonging to a secure partition will be saved on this partition's stack and cleaned by hardware during context switching. Also TF-M cleans secure FP context in FP registers before switching into NSPE to prevent NSPE from probing secure FP context.
CVSS Score	4.3 (Medium)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:C/C:L/I:N/A:N

NS interrupts preempts SPE execution

This section identifies threats on DF5 defined in *Data Flow Diagram*.

Table 21: TFM-GENERIC-NS-INTERRUPT-I-1

Index	**TFM-GENERIC-NS-INTERRUPT-I-1**
Description	Shared registers may contain secure data when NS interrupts occur.
Justification	The secure data in shared registers should be cleaned up before NSPE can access shared registers. Otherwise, secure data leakage may occur.
Category	Information disclosure
Mitigation	On Armv8-M processors with TrustZone, Armv8-M architecture automatically cleans up the registers not banked before switching to Non-secure state while taking NS interrupts. When FPU is enabled in TF-M, with setting of FPCCR_S.TS = 1 besides secure FP context in FP caller registers, FP context in FP callee registers will also be cleaned by hardware automatically when NS interrupts occur, to prevent NSPE from probing secure FP context in FP registers. Refer to Armv8-M Architecture Reference Manual [Arm-ARM] for details. On dual-cpu platforms, shared registers are implementation defined, such as Inter-Processor Communication registers. Dual-cpu platforms must not store any data which may disclose secure information in the shared registers.
CVSS Score	4.3 (Medium)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:C/C:L/I:N/A:N

Table 22: TFM-GENERIC-NS-INTERRUPT-D-1

Index	**TFM-GENERIC-NS-INTERRUPT-D-1**
Description	An attacker may trigger spurious NS interrupts frequently to block SPE execution.
Justification	On Armv8-M processors with TrustZone, an attacker may inject a malicious NS application or hijack a NS hardware to frequently trigger spurious NS interrupts to keep preempting SPE and block SPE to perform normal secure execution.
Category	Denial of service
Mitigation	It is out of scope of TF-M. Assets protected by TF-M won't be leaked. TF-M won't be directly impacted.
CVSS Score	4.0 (Medium)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L

Secure interrupts preempts NSPE execution

This section identifies threats on DF6 defined in *Data Flow Diagram*.

Table 23: TFM-GENERIC-S-INTERRUPT-I-1

Index	**TFM-GENERIC-S-INTERRUPT-I-1**
Description	Shared registers may contain secure data when Armv8-M core switches back to Non-secure state on Secure interrupt return.
Justification	Armv8-M architecture doesn't automatically clean up shared registers while returning to Non-secure state during Secure interrupt return. If SPE leaves critical data in the Armv8-M registers not banked, NSPE can read secure context from those registers and secure data leakage may occur.
Category	Information disclosure
Mitigation	TF-M saves NPSE context in general purpose register R4~R11 into secure stack during secure interrupt entry. After secure interrupt handling completes, TF-M unstacks NSPE context from secure stack to overwrite secure context in R4~R11 before secure interrupt return. Armv8-M architecture will automatically unstack NSPE context from non-secure stack to overwrite other registers not banked, such as R0~R3 and R12, during secure interrupt return, before NSPE software can access those registers. When FPU is enabled in TF-M, with setting of FPCCR_S.TS = 1 and FPCCR_S.CLRONRET = 1, besides secure FP context in FP caller registers, FP context in callee registers will also be cleaned by hardware automatically during S exception return, to prevent NSPE from probing secure FP context in FP registers. Refer to Armv8-M Architecture Reference Manual [Arm-ARM] for details.
CVSS Score	4.3 (Medium)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:C/C:L/I:N/A:N

Miscellaneous threats

This section collects threats irrelevant to the valid TF-M data flows shown above.

Table 24: TFM-GENERIC-STACK-SEAL

Index	**TFM-GENERIC-STACK_SEAL**
Description	Armv8-M processor Secure software Stack Sealing vulnerability.
Justification	On Armv8-M based processors with TrustZone, if Secure software does not properly manage the Secure stacks when the stacks are created, or when performing non-standard transitioning between states or modes, for example, creating a fake exception return stack frame to de-privilege an interrupt, it is possible for Non-secure world software to manipulate the Secure Stacks, and potentially influence Secure control flow. Refer to [STACK-SEAL] for details.
Category	Elevation of privilege
Mitigation	TF-M has implemented common mitigation against stack seal vulnerability. Refer to [ADVISORY-TFMV-1] for details on analysis and mitigation in TF-M.
CVSS Score	5.3 (Medium)
CVSS Vector String	CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:C/C:L/I:L/A:L

Table 25: TFM-GENERIC-SVC-CALL-SP-FETCH

Index	**TFM-GENERIC-SVC-CALL-SP-FETCH**
Description	Invoking Secure functions from handler mode may cause TF-M IPC model to behave unexpectedly.
Justification	On Armv8-M based processors with TrustZone, if NSPE calls a secure function via Secure Gateway (SG) from non-secure Handler mode, TF-M selects secure process stack by mistake for SVC handling. It will most likely trigger a crash in secure world or reset the whole system, with a very low likelihood of over-writing some memory contents.
Category	Denial of service/Tampering
Mitigation	TF-M has enhanced implementation to mitigate this vulnerability. Refer to [ADVISORY-TFMV-2] for details on analysis and mitigation in TF-M.
CVSS Score	4.5 (Medium)
CVSS Vector String	CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:C/C:N/I:L/A:L

Table 26: VLLDM instruction security vulnerability

Index	**TFM-GENERIC-FP-VLLDM**
Description	Secure data in FP registers may be disclosed to NSPE when VLLDM instruction is abandoned due to an exception mid-way.
Justification	Refer to [VLLDM-Vulnerability] for details.
Category	Tampering/Information disclosure
Mitigation	In current TF-M implementation, when FPU is enabled in SPE, TF-M configures NSACR to disable NSPE to access FPU. Therefore, secure data in FP registers is protected from NSPE. Refer to [VLLDM-Vulnerability], for details on analysis and mitigation.
CVSS Score	3.4 (Low)
CVSS Vector String	CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:U/C:L/I:L/A:N

Version control

Table 27: Version control

Version	Description	TF-M version
v0.1	Initial draft	TF-M v1.1
v1.0	First version	TF-M v1.2.0
v1.1	Update version	TF-M v1.5.0
v1.2	Update details to align FP support in NSPE.	TF-M v1.5.0
v1.3	Update for validity of dual-cpu model Armv8-M	TF-M v2.1.0

Reference

Copyright (c) 2020-2024 Arm Limited. All Rights Reserved.

Copyright (c) 2023, Arm Limited. All rights reserved.

Security Advisories

Advisory TFMV-1

Title	NS world may cause the CPU to perform an unexpected return operation due to unsealed stacks.
CVE ID	CVE-2020-16273
Date	16 October 2020
Versions Affected	All versions up to and including TF-M v1.1
Configurations	All
Impact	Most likely a crash in secure world with a very low likelihood of hijacking secure world execution flow via a separate vulnerability.
Fix Version	commit 92e46a3abd0e328fac29ccd1cf161cd482397567
Credit	Matvey Mukha

Background

When the Non-Secure world returns to Secure after a callback (FNC_RETURN) or Non-Secure interrupt handling (EXC_RETURN), the hardware pops the secure return address and RET_PSR from the respective secure stack. The hardware performs a set of integrity checks at this point and will raise a fault if the checks fail. There is a potential vulnerability if the NS attempts a wrong FNC_RETURN or EXC_RETURN and causes the PE to pop from the unexpected stack. Please refer to ARMv8-M Secure stack sealing advisory notice for more details on the vulnerability.

To prevent such an attack, the architecture expects the secure software to *seal* the stacks. The ARMv8-M ARM states that the stack should be sealed with the special value, 0xFEF5EDA5 (informative IGJGL).

Both the MSP_S and the PSP_S stacks need to be sealed to mitigate stack underflow attacks and this is not done currently in TF-M.

Impact

This section analyses the impact of this vulnerability on the upstream TF-M implementation.

All requests coming in from the Non-Secure world uses ARM_LIB_STACK as the PSP_S and then switches to MSP_S as part of SPM scheduling. The MSP_S is fully unwound when the scheduled partition is executing. All partitions run using the PSP_S and this stack can be separate for each partition or be a common stack depending on whether TF-M is in library mode or IPC model. When the partition execution using PSP_S switches to non-secure world due to a non-secure interrupt or a non-secure callback invocation, the non-secure world on return back to secure can use a fake EXC_RETURN or FNC_RETURN operation to trigger an MSP_S stack underflow, and the CPU could fetch the return PC and PSR from the memory just above MSP_S stack (stack grows from higher to lower address).

The memory above MSP_S is the ARM_LIB_STACK (as described by tfm_common_s.sct/ld), which because of overlaying or zero initialization is most likely to be initialized memory for most platforms. This is the stack used to run the initialization code of TF-M and usually there will some unused free space in the stack. Any underflow of MSP_S will be into this stack and hence is likely to be a deterministic crash given that this memory is initialized. In theory, a separate vulnerability that allows an attacker to control the memory above MSP_S in concert with this vulnerability could enable an attacker to hijack the secure world execution flow but the likelihood of that is very low.

Through analysing TF-M specific initialization and execution flow, we have not found any scenario in TF-M in which Non-secure world can fake a return operation back to secure world and cause underflow of PSP_S.

As described in the white paper, de-privileged interrupt handling is also vulnerable to this problem. The Library mode of TF-M uses de-privileged interrupt handling and does not allow non-secure interrupt to pre-empt the secure interrupt handling. But if the de-privileged handler

makes a Non-Secure callback then there is a chance that Non-Secure world could return back to Secure world via a fake FNC_RETURN. Under certain conditions, this can force the CPU to use the 2 words on top of stack as return PC and PSR. Sealing the top of MSP_S before switching to PSP_S as part of de-privileged interrupt handling mitigates this vulnerability.

The interrupt handling in IPC model uses PSA signal to signal the partition and does not use de-privileged interrupt handling mechanism. The PSA signal gets handled by the partition when it is scheduled to run by the SPM. Hence during interrupt handling in IPC model, there is no additional threat caused by this vulnerability, compared to regular partition execution.

Conclusion

Overall, from analysis of upstream TF-M implementation, the severity of this vulnerability is low, given that the values popped out from secure stack (either via underflow of MSP_S or from top of MSP_S stack in de-privileged interrupt handling) cannot be influenced by the Non Secure world. To mitigate the risk completely, the fix in TF-M is to follow the recommendation in ARMv8-M ARM which is to seal the base of both handler mode stack (MSP_S) and thread stacks (PSP_S) and, in case of library mode, also seal the top of MSP_S before handing control over to de-privileged interrupt handler.

Copyright (c) 2020, Arm Limited. All rights reserved.

Advisory TFMV-2

Title	Invoking Secure functions from handler mode may cause TF-M IPC model to behave unexpectedly.
CVE ID	CVE-2021-27562
Date	Mar 5, 2021
Versions Affected	Affected All versions up to and including TF-M v1.2
Configurations	IPC Model on Armv8-M
Impact	Most likely a crash in secure world or reset whole system, with a very low likelihood of overwriting some memory contents.
Fix Version	commit e212ea1637a66255b44d0e7c19ebe9786ab56ccb
Credit	Øyvind Rønningstad, Senior SW Engineer from Nordic Semiconductor

Background

When a non-secure caller calls the secure function, the execution switches the security state via Secure Gateway (SG), then arrived at the TF-M secure entry if the SG is successful. After SG, TF-M invokes SVC to SPM at first because SPM handles requests from SPE and NSPE in a unified SVC handler. The TF-M SVC handler code relies on the 'SPSEL' bit in 'EXC_RETURN' to get the caller stack pointer:

```
; Armv8m mainline as the example
mrs      r2, msp              ; r2 = msp;
tst      lr, #4               ; EXC_RETURN.SPSEL == ?
ite      eq                   ; condition preparation
moveq    r0, r2               ; if (EXC_RETURN.SPSEL == 0) r0 = msp;
movne    r0, psp              ; if (EXC_RETURN.SPSEL == 1) r0 = psp;
mov      r1, lr               ; r1 = EXC_RETURN;
bl       tfm_core_svc_handler ; r0 = sp(context), r1 = EXC_RETURN, r2 = msp
```

If NSPE calls the secure function in 'Handler mode', the TF-M secure entry runs in 'Handler mode' before invoking SVC because PE mode does not change during the transition from non-secure state to secure state via the successful SG. Main stack pointer (MSP) is always used in 'Handler

mode', which is irrelevant to the value of SPSEL. However, the code above selects secure process stack pointer (PSP_S, S suffix here indicates the security state) for further SVC handling based on EXC_RETURN.SPSEL, hence the SVC handler accesses the wrong stack for handling the request in the NSPE caller in 'Handler mode' case. To prevent this vulnerability, TF-M secure SVC handler should fetch the right stack pointer register by checking both the PE mode and SPSEL bit. The handler should also prevent callers in non-secure *Handler mode* from calling TF-M SVC functionalities. The following section analysis impact of the IPC model. Library model is not vulnerable to this attack because it checks the PE mode in the TF-M secure entry.

Impact

When the vulnerability is happening, PSP_S is the incorrect stack pointer fetched, the impact is decided by the content PSP_S is pointing to. The SVC handler gets the SVC number by referencing the memory at the 2 bytes subtracted position of member 'Return Address' in the PSP_S pointing context, and uses the caller saved registers in the context as parameters for the subsequent functions indicated by the SVC number. The PSP_S may point to the bottom (higher address) of secure thread stack if there is no ongoing secure function call, or it points to a preempted context caused by non-secure preempting secure execution. When PSP_S is pointing to the stack bottom when this issue happens, the caller context is pointing to the underflowed stack of which the top 2 words are stack seal, and the rest of the context is unpredictable. These underflowed content are ZERO initialized for most platforms, hence when fetching the SVC Number at memory address 'Return Address - 2', a 'MemFault' or 'HardFault' would happen. Even if a valid SVC number is returned, the stack seal values are part of the parameters for the subsequent functions that have parameters, which cannot pass the validation for the parameters. When PSP_S is pointing to a preempted context, the preempted place can be the TF-M secure entry area or internal thread space. If the preemption happens when the execution is in the TF-M secure entry area, the PSP_S will point to an NS preemption stack frame and hence will fail the context size validation through this entry into TF-M SVC Handler. In the other scenario that TF-M internal thread is preempted, there is a very remote chance that the exception stack frame will contain a context with a valid SVC Number with valid parameters. As a summary, the impacts of the vulnerability depend on whether an SVC number could be fetched from the incorrect stack and what operation the fetched SVC number corresponds to:

- A memory access fault when fetching the SVC number from memory. This fault halts the whole system. A reset can recover the system back to normal.
- The SVC number corresponds to the initialization process in TF-M. SPM initialization is called for the second time. This re-initialization process will fail and halt the whole system. A reset can recover the system back to normal.
- The SVC number corresponds to the boot data fetching function. If the unpredictable data in stack triggers boot data fetching and passes the parameter validation under a very rare case, boot data is copied into an unpredictable memory address. One note here, the TF-M default boot data has no sensitive information.
- The SVC number corresponds to the log output function. If the unpredictable data in stack triggers log output under a very rare case, secure data at unpredictable address might be printed out through the log interface.
- The SVC number corresponds to other valid numbers other than above. The system resets because of parameter validation fail.

Conclusion

Overall, from the analysis of upstream TF-M implementation, the severity of this vulnerability is Medium, given that a software crash or system reset happen most of the time. The probability for causing secure data leakage via log output or tampering of secure memory with boot data is extremely low as the TF-M internal thread exception stack frame contents have to pass all the validations performed by SPM.

Copyright (c) 2021, Arm Limited. All rights reserved.

Advisory TFMV-3

Title	abort() function may not take effect in TF-M Crypto multi-part MAC/hashing/cipher operations.
CVE ID	CVE-2021-32032
Public Disclosure Date	May 10, 2021
Versions Affected	Affected all versions up to and including TF-M v1.3.0
Configurations	All
Impact	It can cause memory leakage in TF-M Crypto service, eventually making TF-M Crypto service unavailable and impacting other services relied on it.
Fix Version	commit 7e2e52
Credit	Chongqing Lei, Southeast University Zhen Ling, Associate Professor, Southeast University Xinwen Fu, Professor, University of Massachusetts Lowell

Background

PSA multi-part crypto operation sequence

PSA Crypto API specification defines a common sequence for all multi-part crypto operations. The sequence can be simplified to the following steps:

- setup() sets up the multi-part operation.
- update() adds data/configurations into the multi-part operation.
- finish() completes the multi-part operation.

PSA Crypto API specification requests that the corresponding abort() function shall be called when update() or finish() function fails. The abort() function aborts the ongoing multi-part operation and cleans up the operation context.

TF-M multi-part crypto operation functions eventually call the underlying crypto library (Mbed TLS by default) to perform those steps, including abort() step.

PSA multi-part crypto operation objects

PSA Crypto API specification defines an operation object for each type of multi-part crypto operations. For example, psa_mac_operation_t for multi-part MAC operations and psa_hash_operation_t for multi-part hashing operations.

TF-M Crypto service relies on the underlying crypto library (Mbed TLS by default) to implement those objects. The structures of those objects are crypto library specific and hidden to TF-M. The underlying crypto library usually stores and manages the context of ongoing multi-part crypto operations in the corresponding PSA operation object. For example, Mbed TLS stores multi-part hashing operation context in its psa_hash_operation_t implementation.

The context is cleaned up in crypto library abort() function when the client calls abort() to handle a previous error. The clean-up execution can include zeroing the memory area and freeing allocated memory.

TF-M multi-part crypto operation objects

TF-M Crypto service defines a dedicated operation structure tfm_crypto_operation_s to wrap PSA multi-part crypto operation object and maintains its own status, as shown in the code block below.

```
struct tfm_crypto_operation_s {

    ...

    union {
        psa_cipher_operation_t cipher;     /*!< Cipher operation context */
        psa_mac_operation_t mac;           /*!< MAC operation context */
        psa_hash_operation_t hash;         /*!< Hash operation context */
        psa_key_derivation_operation_t key_deriv; /*!< Key derivation operation context */
    } operation;
};
```

TF-M Crypto service assigns a tfm_crypto_operation_s object for each multi-part crypto operation sequence during setup() step. The tfm_crypto_operation_s object content will be cleaned after the sequence completes or fails.

Impact

During multi-part hashing/MAC/cipher operations, if the underlying crypto library function returns an error code, TF-M update() and finish() functions will immediately clean up the structure tfm_crypto_operation_s content and exit.

When tfm_crypto_operation_s content is cleaned in TF-M update() and finish() functions, the content in PSA multi-part crypto operation object inside tfm_crypto_operation_s is also cleaned. If the underlying crypto library stores operation context in the PSA operation object, the operation context is lost before clients call abort() to handle the error.

Therefore, the underlying crypto library abort() function can be unable to perform normal abort operation if it cannot fetch the context or its content. In other words, the underlying crypto library abort() may not work normally or take effect.

In theory when the case analyzed above occurs:

- If the underlying crypto library dynamically allocates some memory regions during multi-part operation and stores those memory region pointers in the PSA multi-part operation object, the underlying crypto library will be unable to locate and free those allocated memory regions in abort(). It will cause memory leakage in TF-M Crypto service. It may further make TF-M Crypto service unavailable and affect other services relying on TF-M Crypto service.
- The underlying crypto library abort() may still consider the field values in the context as valid. abort() may perform unexpected behaviors or access invalid memory regions. It may trigger further faults and block TF-M Crypto service or even the whole system.

Note: The actual consequences depend on the implementation of the multi-part operations in the underlying crypto library.

Impacted PSA Crypto API functions

The following PSA multi-part crypto operation functions are impacted:

- Multi-part hashing operations
  - psa_hash_update()
  - psa_hash_finish()
  - psa_hash_verify()
  - psa_hash_clone()
- Multi-part MAC operations
  - psa_mac_update()
  - psa_mac_sign_finish()
  - psa_mac_verify_finish()
- Multi-part cipher operations
  - psa_cipher_generate_iv()
  - psa_cipher_set_iv()
  - psa_cipher_update()
  - psa_cipher_finish()

Justifications on unaffected multi-part operations

TF-M multi-part AEAD operations and multi-part key derivation operations are not impacted by this issue.

TF-M Crypto service has not implemented multi-part AEAD operations. TF-M multi-part AEAD functions directly return an error of unsupported operations.

In TF-M key derivation implementation, the psa_key_derivation_operation_t object is only cleaned in the abort() function after the underlying crypto library completes abort.

Mitigation

The clean-up operation shall be removed from error handling routines in the following TF-M Crypto functions:

- Multi-part hashing operations
  - tfm_crypto_hash_update()
  - tfm_crypto_hash_finish()
  - tfm_crypto_hash_verify()
  - tfm_crypto_hash_clone()
- Multi-part MAC operations
  - tfm_crypto_mac_update()
  - tfm_crypto_mac_sign_finish()
  - tfm_crypto_mac_verify_finish()
- Multi-part cipher operations
  - tfm_crypto_cipher_generate_iv()
  - tfm_crypto_cipher_set_iv()
  - tfm_crypto_cipher_update()
  - tfm_crypto_cipher_finish()

Note: This mitigation assumes that client follows the sequence specified in PSA Crypto API specification to call abort() when an error occurs during multi-part crypto operations.

Copyright (c) 2021, Arm Limited. All rights reserved.

Advisory TFMV-4

Title	NSPE may access secure keys stored in TF-M Crypto service in Profile Small with Crypto key ID encoding disabled.
CVE ID	CVE-2021-40327
Public Disclosure Date	22nd Nov, 2021
Versions Affected	TF-M v1.4.0
Configurations	Profile Small
Impact	In Profile Small, secure keys stored in Crypto service can be leaked to NSPE if NSPE acquires secure key IDs.
Fix Version	Commit 42e77b and v1.4.1
Credit	N/A

Background

TF-M Profile Small disabled Crypto key ID encoding with key owner client ID in TF-M v1.4.0 release.

When the Crypto key is stored into TF-M Crypto service, the key ID is not encoded with the client ID of key owner in Profile Small in TF-M v1.4.0. Therefore, TF-M Crypto service is unable to distinguish or validate owners of keys in Profile Small. NSPE can access the keys belonging to SPE in Profile Small in some scenarios.

Details

In TF-M v1.4.0, TF-M Crypto service by default relies on two mechanisms to validate key owners in key management.

- TF-M Crypto service maintains a key handle array. When a key is stored in Crypto service, the key ID and the key owner client ID are stored in the array. When a caller requests to access a key, TF-M Crypto service validates the request by comparing the caller client ID with the stored key client ID.
- Mbed TLS stores a special structure encoded by key owner client ID and the key ID. When a caller requests to access a key, Mbed TLS validates the request by comparing the caller client ID with the key client ID stored in that structure.

Secure clients are not isolated from each other in Profile Small and it doesn't require to validate key owner client ID between secure clients. Therefore, in TF-M v1.4.0, Profile Small disabled both mechanisms above to optimize the key storage size. The key directly or indirectly stored via psa_import_key() is not encoded with key owner client ID.

However, it also disables the validation of NS client ID when a NS client accesses keys stored in TF-M Crypto. NS clients can call psa_open_key()/psa_export_key() to access secure clients' keys stored via psa_import_key(), if NS clients acquire the key ID of secure clients.

Impact

Only TF-M Profile Small is impacted. All the other configurations or Profiles are not affected.

Analysis of RoT services in Profile Small

TF-M Profile Small enables Internal Trusted Storage (ITS), Crypto and Initial Attestation by default. The following analysis focuses on the impact on RoT services in Profile Small.

- ITS service doesn't create or store its own key in Crypto service. It is not impacted directly.
- Crypto service key derivation may be impacted.
  - psa_key_derivation_output_key() eventually stores the derived key in Crypto service. The stored derived keys can be accessed by a NS client if the NS client acquires the derived key ID value.
  - Platform specific implementation may store Hardware Unique Key (HUK) into Crypto service for key derivation from HUK via psa_import_key().
    * Platform driver may import HUK as a temporary key into Crypto service during derivation and close the temporary key when derivation completes.

      If a NS client preempts the derivation and calls PSA Cryptography API to access temporary HUK data stored in Crypto service, the access will be captured by TF-M re-entry detection and rejected by TF-M SPE.
    * Platform driver may permanently store HUK via Crypto service for derivation and the key is still managed by Crypto service when NSPE is running.

      NS client can access HUK data via PSA Cryptography API if it acquires the key ID of stored HUK.
- Symmetric key algorithm based Initial Attestation temporarily stores symmetric Initial Attestation Key (IAK) in Crypto service during Initial Attestation Token generation. It imports symmetric IAK into Crypto service during generation and removes it from Crypto service when generation completes.

  If a NS client preempts the generation and calls PSA Cryptography API to access the temporary IAK data stored in Crypto service, the access will be captured by TF-M re-entry detection and rejected by TF-M SPE.

  Therefore, Initial Attestation is not impacted directly.

Impact on Profile Small default implementation

Default Profile Small RoT services don't initially call Crypto key derivation or store any secure key into Crypto service.

According to the analysis of RoT services above, device HUK can be accessed by NS clients and leaked to NSPE, in Profile Small default implementation, when all the following conditions are met.

- Platform specific implementation stores HUK in Crypto service, initially or during a derivation requested by NS client.
- HUK is still stored in Crypto service when NSPE is running.
- An NS client acquires the key ID of HUK in Crypto service and accesses HUK key via PSA Cryptography API.

Other vulnerabilities are not found yet so far.

Impact on vendor RoT services

If a vendor RoT service is integrated in Profile Small, its keys stored via psa_import_key() or derived from psa_key_derivation_output_key() can be accessed by NS client and leaked to NSPE when both following conditions are met.

- The secure key is stored in Crypto service when NSPE is running.
- An NS client acquires the key ID and accesses the key via PSA Cryptography API.

How NS client can acquire secure key ID is related to key management implementation of the underlying crypto library in TF-M Crypto service. With default Mbed TLS, NS hackers can import a NS key at first to obtain the rough base value of Mbed TLS key slots and then try a smaller subset of key ID values by brute-force.

Mitigation

This issue has been fixed by enforcing Mbed TLS key ID encoding with key owner client ID to be enabled.

This patch intended to optimize TF-M Crypto service key handle array and coincidentally fixed the issue.

v1.4.1 fixed this issue as a patch release.

Copyright (c) 2021, Arm Limited. All rights reserved.

Advisory TFMV-5

Title	psa_fwu_write() may cause buffer overflow in SPE.
CVE ID	CVE-2021-43619
Public Disclosure Date	Feb 11, 2022
Versions Affected	From 3e7129f to 921d0ea
Configurations	IPC model with Firmware Update partition enabled
Impact	In IPC model, the caller of psa_fwu_write() from SPE or NSPE can overwrite the stack memory outside of the local buffer in Firmware Update partition.
Fix Version	commit 78f7530
Credit	Mark Horvath, Staff Software Engineer from Arm Ltd.

Background

In Firmware Update partition, the psa_fwu_write() service is declared as:

```
/**
 * \brief Writes an image to its staging area.
 *
 * Writes the image data 'block' with length 'block_size' to its staging area.
 *
 * \param[in] image_id        The identifier of the image
 * \param[in] block_offset    The offset of the block being passed into block,
 *                            in bytes
 * \param[in] block           A buffer containing a block of image data. This
 *                            might be a complete image or a subset.
 * \param[in] block_size      Size of block. The size must not be greater than
 *                            PSA_FWU_MAX_BLOCK_SIZE.
 *
 */
psa_status_t psa_fwu_write(psa_image_id_t image_id,
                           size_t block_offset,
                           const void *block,
                           size_t block_size);
```

In IPC model, this service calls the tfm_fwu_write_ipc() API to write the input data into the device. In this API, the block_size bytes input data (in block argument) is read into a 1024 bytes local buffer

via the psa_read() API. If the input argument block_size is greater than 1024, then the memory space starting from the address of the local buffer with block_size bytes would be overwritten by the input data in block argument.

Impact

In IPC model, the caller of psa_fwu_write() from SPE or NSPE can overwrite the memory space in RAM. The overwritten memory space ranges from the address of the local buffer which locates at the stack of Firmware Update partition to the end of the RAM. The overwritten memory may include the data of SPM and device drivers, as well as part of the stack of Firmware Update partition.

Mitigation

Add check against the input buffer length in the tfm_fwu_write_ipc() API before reading the input data into the local buffer. See commit 78f7530.

Copyright (c) 2022, Arm Limited. All rights reserved.

Advisory TFMV-6

Title	Partial tag comparison when using Chacha20-Poly1305 on the PSA driver API interface in CryptoCell enabled platforms
CVE ID	CVE-2023-40271
Public Disclosure Date	04/09/2023
Versions Affected	TF-M v1.6.0, TF-M v1.6.1, TF-M v1.7.0, TF-M v1.8.0
Configurations	CC312 enabled platforms, where the legacy driver API is disabled (CC312_LEGACY_DRIVER_API_ENABLED=OFF) and the single part AEAD APIs are implemented through a dedicated function and not by leveraging the multipart functions (CC3XX_CONFIG_ENABLE_AEAD_ONE_SHOT_USE_MULTIPART not set)
Impact	It might allow for unauthenticated payloads to be deemed as authentic by comparing only the first 4 bytes of the authentication tag instead of the full length of 16 bytes
Fix Version	2e82124af, TF-M v1.8.1
Credit	Nordic Semiconductor

Background

AEAD algorithms

Authenticated Encryption with Associated Data (AEAD) is a common ciphering method where the data to be encrypted is also authenticated as part of the process by creating an authentication tag. When the encrypted data is then decrypted, the authentication tag is verified and if it does not match the expected value, then the entire operation fails. In this way, the operation allows

for Authenticity in addition to the confidentiality granted by the encryption process. Using PSA Crypto APIs, it's possible to use several of such algorithms, such as AES in Galois-Counter Mode (GCM), AES in Counter with CBC-Mac Mode (CCM) or Chacha20-Poly1305, which is a combination of the Chacha20 cipher with the Poly1305 authenticator[1].

In particular for Chacha20-Poly1305 the corresponding macro defining the algorithm in PSA Cryptographic API specification is PSA_ALG_CHACHA20_POLY1305.

Single part vs multipart API functions

PSA Crypto API specification[2] allows the usage of AEAD algorithms through several possible APIs, that can be grouped generally in the single part, or integrated, operation type, and in the multi-part operation type. The main difference is that for single part operations, the whole encryption and tag production (and on the other hand, the whole decryption and tag authentication) happen with a single API call. This type of approach is simpler but at the same time less flexible than the multipart approach where an operation context must be allocated by the application and the encryption/decryption processes require the call to different APIs to setup the operation context, provide inputs to the process and calculate the output. For example, the integrated APIs defined by the PSA Crypto API spec are called psa_aead_encrypt() and psa_aead_decrypt.

It is possible, to reduce the code size of an implementation, to implement the single part APIs by calling the underlying multipart functions, effectively encapsulating the multipart flow in the single part APIs.

PSA Unified Driver API for Cryptoprocessors

The PSA Unified Driver API for Cryptoprocessors spec[3] allows a PSA compliant implementation to redirect some of the operations to the underlying hardware accelerated platform. For example, CryptoCell is a cryptographic accelerator available in some of the TF-M supported platforms that can accelerate crypto operations, such as Chacha20-Poly1305. The driver code associated to it has the responsibility of driving the hardware resources with inputs and collect the outputs. When decrypting, the driver must compare the reconstructed authentication tag with the expected value, and return failure in case of mismatch during verification. For Chacha20-Poly1305, the tag size is 16 bytes.

For CryptoCell enabled platforms, the software component implementing the PSA Unified Driver interface is located into the psa_driver_api directory in lib/ext/cryptocell-312-runtime/codesafe/src/ in TF-M's codebase. This components can be configured to have single part APIs implemented though the corresponding multipart functions by setting the following define at build time: CC3XX_CONFIG_ENABLE_AEAD_ONE_SHOT_USE_MULTIPART.

By default, CryptoCell enabled platforms don't build the PSA Unified Driver API interface layer but rely on the legacy interface. To enable the PSA Driver interface the following TF-M build option during CMake config must be be set to OFF: CC312_LEGACY_DRIVER_API_ENABLED=OFF.

Impact

When the PSA Driver API interface is used and is not configured to rely on the corresponding multipart functions, when performing the verification of the authentication tag at the end of the authenticated decryption process, the buffer containing the tag is only partially verified (the first 4 bytes only instead of the full 16 bytes). This allows for the possibility of unauthenticated data to be recognized as authentic. An attacker could theoretically construct a malicious payload to actively exploit such partial verification.

Impacted PSA Crypto API functions

[1] Chacha20 and Poly1305 for IETF Protocols: https://datatracker.ietf.org/doc/html/rfc7539
[2] PSA Cryptographic API v1.1: https://armmbed.github.io/mbed-crypto/html/
[3] PSA Unified Driver interface: https://github.com/Mbed-TLS/mbedtls/blob/development/docs/proposed/psa-driver-interface.md

The following PSA single part crypto operation function is impacted:

- psa_aead_decrypt

Mitigation

The verification of the authentication tag must happen on the full 16 bytes of instead of just the first 4 bytes. This means that loop that currently performs such verification in the cc3xx_decrypt_chacha20_poly1305() function must be changed from this:

```
/* Check tag in "constant-time" */
for (diff = 0, i = 0; i < sizeof(tag_length); i++)
    diff |= tag[i] ^ local_tag_buffer[i];
```

to this:

```
/* Check tag in "constant-time" */
for (diff = 0, i = 0; i < tag_length; i++)
    diff |= tag[i] ^ local_tag_buffer[i];
```

References

Copyright (c) 2023, Arm Limited. All rights reserved.

Advisory TFMV-7

Title	ARoT can access PRoT data via debug logging functionality
CVE ID	CVE-2023-51712
Public Disclosure Date	The issue was publicly reported on 2023.12.04
Versions Affected	All version up to TF-M v2.0.0 inclusive
Configurations	IPC mode with TFM_SP_LOG_RAW_ENABLED=1
Impact	A malicious ARoT partition can expose any part of memory via stdio interface if TFM_SP_LOG_RAW_ENABLED is set
Fix Version	TBD
Credit	Roman Mazurak, Infineon

Background

TF-M log subsystem if enabled by TFM_SP_LOG_RAW_ENABLED config option, uses a SVC call to print logging messages on the stdio output interface. Since the SVC handler has the highest privilege level and full memory access, this communication channel can be exploited to expose any memory content to stdout device, usually UART. The logging subsystem is available to the secure side only but in isolation level 2 and higher PSA Root of Trust partitions (PRoT) shall be protected from an access from Application Root of Trust (ARoT) partitions. Although a direct call of tfm_hal_output_sp_log() from ARoT partition will be blocked by MPU raising the MemoryManagement() exception, a malicious ARoT partition can create an alternative SVC call to output any memory data like this:

```
static int tfm_output_unpriv_string(const unsigned char *str, size_t len)
{
    __ASM volatile("SVC %0          \n"
                   "BX LR           \n"
                   : : "I" (2));
}
```

Impact

In IPC mode with PSA isolation level 2 and higher and TFM_SP_LOG_RAW_ENABLED option enabled an ARoT partition can expose to the stdout device any memory data using TF-M logging subsystem via SVC call.

Mitigation

Ensure that data sent for logging belongs to the current partition. For that purpose tfm_hal_memory_check(curr_partition->boundary, data, size, TFM_HAL_ACCESS_READABLE) is added to the logging function of the SVC handler. If the check fails then tfm_core_panic() is invoked and system halts.

Copyright (c) 2024, Arm Limited. All rights reserved.

Advisory TFMV-8

Title	Unchecked user-supplied pointer via mailbox messages may cause write of arbitrary address.
CVE ID	CVE-2024-45746
Public Disclosure Date	October 02, 2024
Versions Affected	All version from TF-Mv1.6.0 up to TF-Mv2.1.0 inclusive
Configurations	Platforms with standard mailbox dispatcher tfm_spe_mailbox.
Impact	The mailbox message could contain arbitrary pointers which, in case of psa_call failure, would lead to write to a user-specified adddress in memory.
Fix Version	5ae0a02e8 TF-M v2.1.1
Credit	Infineon Technologies AG, in collaboration with: Tobias Scharnowski, Simon Wörner and Johannes Willbold from fuzzware.io.

Background

The psa_call message through the mailbox contains input/output vectors along with their respective lengths. This message is provided by a NSPE client. SPE takes the message and pass it to the mailbox dispatcher (tfm_spe_mailbox), which handles the message by performing a copy of the i/o vectors into local arrays. When either the client_id translation or the psa_call fails, the dispatcher replies immediately to the client. At that moment, the outvec is written back for its given length, which may not have been sanitized beforehand, resulting in arbitrary access of memory if the provided length goes beyond the legit vector size.

Impact

When the dispatcher in tfm_spe_mailbox is used, a user through mailbox could write into arbitrary address by first placing the malicious data into the local vectors with a bad message, then subsequently sending a psa_call with an invalid vector length. If both calls fail, the reply routine in tfm_spe_mailbox could take the injected data and write it into a desired location specified by the invalid length. Note that the above sequence would require sending the two mesages through two different mailbox slots.

Mitigation

Ensure that the outvec is written back only when the psa operation is successful. Any errors ahead of replying must be taken as a hint to avoid such write-back since they may be due to

wrong supplied user-data in the vectors (pointers, length etc). To achieve the above, proper sanitization of input data must also be performed and related errors propagated to the reply subroutine.

Copyright (c) 2024, Arm Limited. All rights reserved.

Advisory TFMV-9

Title	FWU does not check the length of the TLV's payload
CVE ID	CVE-2025-53022
Public Disclosure Date	Jul 21, 2025
Versions Affected	All version from TF-M v1.3.0 up to TF-M v2.2.0 inclusive
Configurations	Any with Firmware Update (FWU) partition enabled
Impact	A Type-Length-Value (TLV) payload can be larger than the image dependency resulting in out-of-bounds write
Fix Version	fab8ccd32940ea1a7e9f38bc5f4e4e55c25afc44
Credits	Bartek Piekarski Distinguished Engineer, Product Security, Arm Ltd.

Background

In Firmware Update partition, the fwu_bootloader_install_image() function checks the authenticity and integrity of the image candidates. For each image candidate, its dependencies are loaded and when criteria are met, the candidate image is staged. During the dependencies check, the length of the image dependency fetched from the TLV is not verified, and this could lead to overflow in read and write from memory in subsequent steps.

Impact

An incorrect and unchecked TLV data can result in accessing the reading and writing memory for the candidate images that goes beyond the bounds, possibly causing in a redirection of program execution.

Mitigation

Add a length check in the fwu_bootloader_install_image() immediately after the next TLV is found by bootutil_tlv_iter_next(), during the dependencies fetch. See commit fab8ccd32940ea1a7e9f38bc5f4e4e55c25afc44.

SPDX-License-Identifier: BSD-3-Clause

SPDX-FileCopyrightText: Copyright The TrustedFirmware-M Contributors

ID	Title
TFMV-1	NS world may cause the CPU to perform an unexpected return operation due to unsealed stacks.
TFMV-2	Invoking Secure functions from handler mode may cause TF-M IPC model to behave unexpectedly.
TFMV-3	abort() function may not take effect in TF-M Crypto multi-part MAC/hashing/cipher operations.
TFMV-4	NSPE may access secure keys stored in TF-M Crypto service in Profile Small with Crypto key ID encoding disabled.
TFMV-5	psa_fwu_write() may cause buffer overflow in SPE.
TFMV-6	Partial tag comparison when using Chacha20-Poly1305 on the PSA driver API interface in CryptoCell enabled platforms
TFMV-7	ARoT can access PRoT data via debug logging functionality
TFMV-8	Unchecked user-supplied pointer via mailbox messages may cause write of arbitrary address
TFMV-9	FWU does not check the length of the TLV's payload

SPDX-License-Identifier: BSD-3-Clause

SPDX-FileCopyrightText: Copyright The TrustedFirmware-M Contributors

Security Disclosures

Trusted Firmware-M(TF-M) disclose all security vulnerabilities, or are advised about, that are relevant to TF-M. TF-M encourage responsible disclosure of vulnerabilities and try the best to inform users about all possible issues.

The TF-M vulnerabilities are disclosed as Security Advisories, all of which are listed at the bottom of this page.

Found a Security Issue?

Although TF-M try to keep secure, it can only do so with the help of the community of developers and security researchers.

Warning: If any security vulnerability was found, please **do not** report it in the issue tracker or on the mailing list. Instead, please follow the Security incident process.

One of the goals of this process is to ensure providers of products that use TF-M have a chance to consider the implications of the vulnerability and its remedy before it is made public. As such, please follow the disclosure plan outlined in the Security Incident Process. TF-M do the best to respond and fix any issues quickly.

Afterwards, write-up all the findings about the TF-M source code is highly encouraged.

Attribution

TF-M values researchers and community members who report vulnerabilities and TF-M policy is to credit the contributor's name in the published security advisory.

Copyright (c) 2020-2023, Arm Limited. All rights reserved.

Releases

Version 2.1.4

LTS Support

- Upgrade to MbedTLS v3.6.5, which contains security advisories[1].

Reference

SPDX-License-Identifier: BSD-3-Clause

SPDX-FileCopyrightText: Copyright The TrustedFirmware-M Contributors

Version 2.1.3

LTS Support

- Cherry-picks for vulnerability and critical bugs fixed in MCUboot v2.2.0
- FWU: check the TLV payload length sanity
- internal_trusted_storage: allow tfm_its_get_plain to be compiled
- platform: adi: max32657: Adding ITS Encryption support for upsteam TF-M
- platform: stm: fix syntax error for rsa 2048 key otp provisioning
- Upgrade to MbedTLS v3.6.4-LTS, which carries fixes from v3.6.3.
- secure_fw: partitions: crypto: Add missing psa apis to crypto_spe.h
- mcuboot: Fix SWAP_USING_MOVE FWU method
- platform: adi: fix flash driver, enable PS, mbedtls TRNG, s_ns_access.cmake
- STM : fix build when BL2 is disable
- Crypto: Add psa_can_do_cipher()

New security advisories

A new security vulnerability has been fixed in v2.1.3. Refer to *TFMV-9* for more details. The mitigation is included in this release.

SPDX-License-Identifier: BSD-3-Clause

SPDX-FileCopyrightText: Copyright The TrustedFirmware-M Contributors

Version 2.1.2

New features

- Mbed TLS upgrade to v3.6.3.

New platforms supported

- *MAX32657*

[1] Mbed TLS 3.6.5

Hotfixes

- Protected Storage content can be lost if the storage is “full”, psa_ps_set is attempted and device is reset (twice).
- Fix wrapper to properly mark NSPE images as such
- SPM: Remove specific section for psa_interface_thread_fn_call
- SPM: interrupts: Add missing checks on fih_rc return value
- SPM: Services do not unmap IOVECS
- SPM: SPM does not return PSA_ERROR on refused psa_connect
- SPM: Fixes for invalid connection retention after psa_close
- SPM: Fixes for reverse-handle

Known issues

Some open issues are not fixed in this release.

Descriptions	Issue links
SPM does not automatically unmap mm-iovecs.	https://github.com/TrustedFirmware-M/trusted-firmware-m/issues/20

SPDX-License-Identifier: BSD-3-Clause

SPDX-FileCopyrightText: Copyright The TrustedFirmware-M Contributors

Version 2.1.1

New features

- Mbed TLS upgrade to v3.6.2.

New security advisories

A new security vulnerability has been fixed in v2.1.1. Refer to *TFMV-8* for more details. The mitigation is included in this release.

New platforms supported

- *RP2350.*

Hotfixes

- Crypto: Additional checks for writes to avoid out-of-bound access
- Crypto: Prevent the scratch allocator from overflowing
- Crypto: Protect writes to avoid out-of-bound access
- SPM: mailbox_agent_api: Free connection if params association fails

---

SPDX-License-Identifier: BSD-3-Clause SPDX-FileCopyrightText: Copyright The TrustedFirmware-M Contributors

Version 2.1.0

New major features

- TF-M aligns the Crypto service to the same PSA Crypto headers used by the Mbed TLS 3.6.0 reference implementation
  - Refer to the *TF-M Crypto service design document* for a detailed description of the firmware architecture of the service.
- Initial support for on-core and off-core clients on Hybrid platforms (A-profile + M-profile or M-profile + M-profile) using solution 1 as described in[1],[2]. The functionality is still under active development.
- P256-M[3] component is enabled on the BL2 stage for image signature verification based on ECDSA.
- MCUboot upgrade to v2.1.0.
- Mbed TLS upgrade to v3.6.0.
- BL2 now provides a *thin* PSA Crypto core layer when MCUBOOT_USE_PSA_CRYPTO=ON and can use builtin keys when ECDSA based signature verification is selected with MCUBOOT_SIGNATURE_TYPE="EC-P256".

New security advisories

A new security vulnerability has been fixed in v2.1.0. Refer to *TFMV-7* for more details. The mitigation is included in this release.

New platforms supported

- *Alcor (AN557).*
- *Corstone-315.*

Tested platforms

The following platforms are successfully tested in this release.

- **Arm**
  - AN519
  - AN521
  - AN555
  - Corstone-300
  - Corstone-310
  - Corstone-315
  - Corstone-1000
  - Musca-B1
  - Musca-S1
- **ArmChina**
  - Alcor (AN557)
- **STM**

[1] TF-M Hybrid Platform Demo, TF-M tech forum 11-04-2024
[2] Trusted Firmware-M and Hybrid platforms, TF-M tech forum 14-09-2023
[3] P256-M

  - NUCLEO-L552ZE-Q
  - STM32H573idk
- **Infineon/Cypress**
  - PSoC 64
- **NXP**
  - LPCXpresso55S69

Reference memory footprint

All measurements below are made for *AN521* platform, built TF-Mv2.1.0-RC2 on Windows 10 using Armclang v6.18 and build type MinSizeRel.

All modules are measured in bytes. Some minor modules are not shown in the table below.

Note: Profile *Medium-ARoT-less* built with disabled Firmware Update service to align with other TF-M Profiles.

Module	Base		Small		ARoT-less		Medium		Large	
	Flash	RAM	Flash	RAM	Flash	RAM	Flash	RAM	Flash	RAM
Generated	112	3184	160	3184	160	3184	208	3184	272	3184
Objects	972	1056	1282	5444	1379	6128	1517	1468	1588	1468
c_w.l	190	0	568	0	568	0	568	0	808	0
platform_s.a	5142	288	5474	288	5826	288	6198	288	6328	288
spm.a	3640	173	4522	173	4012	173	6616	1385	6782	1390
sprt.a	274	0	1438	0	1284	0	2438	4	2418	4
mbedcrypto.a	0	0	25588	2108	30104	2104	30104	2104	78012	1988
PROT_attestation.a	0	0	2341	557	2571	1218	2571	3010	2687	3010
PROT_crypto.a	0	0	3336	2046	3846	16002	3846	22914	4318	25794
PROT_its.a	0	0	4830	80	4894	112	5064	1988	5068	2468
PROT_platform.a	0	0	0	0	486	0	526	1280	526	1280
AROT_ps.a	0	0	0	0	0	0	3280	4364	3280	4364
Padding	34	35	113	44	114	15	120	47	171	38
plat-form_crypto_keys.a	0	0	246	0	252	0	252	0	252	0
qcbor.a	0	0	854	0	854	0	854	0	854	0
crypto_service_p256ı	0	0	0	0	3534	0	3534	0	0	0
Total inc. Padding	10364	4736	50752	13924	59884	29224	67696	42036	113364	45276

Known issues

Some open issues are not fixed in this release.

Descriptions	Issue links
TF-M Kconfig is broken due to build split. It will be recovered in a future release.	Not tracked
The message rhandle is overridden in the backend for ns_agent_mailbox. PSA ACK tests in IPC mode on platforms using ns_agent_mailbox fail for this reason.	Not tracked

Issues fixed since v2.0.0

The following issues have been fixed since the v2.0.0 release.

Descriptions	Issue links
<None>	<None>

Reference

Copyright (c) 2024, Arm Limited. All rights reserved.

Version 2.0.0

New major features

- TF-M secure build process and non-secure build process are split to simplify and enhance non-secure integration with TF-M.
  - Refer to *TF-M Build Instruction* to build SPE image.
  - Refer to *Building Tests* to build non-secure tests.
- *Update new Mailbox agent API*.
  - Decouple the specific application Mailbox from SPM, make it an application in Secure Partition.
  - Unify the interfaces between partitions and SPM, and reduces the interaction interface between them.
- Multi-core support in the Secure Function (SFN) model.
- Optimize SPM critical section implementation to reduce time cost in isolation level 2&3.
- Use local variables for connection handles instead of dynamic allocation when there is only synchronous service access routine in the SFN model.
- P256-M[1] component is enabled on the TF-M side in profile medium which has a much smaller code size and RAM footprint.
- MCUboot upgrade to v2.0.0.
- Mbed TLS upgrade to v3.5.0.
- TF-M PSA client API performance profiling is tracked in SQUAD[2] and the profiling tool[3] is updated.
- TF-M integrates Read the Docs[4] to support finding documentation versions by the released tags and downloading PDFs. External links are supported for documentation in TF-M Tests, Tools and Extras repositories.

New security advisories

A Security vulnerability fixed in v1.8.1 Refer to *TFMV-6* for more details. The mitigation is included in this release.

New platforms supported

- *stm32h573i_dk.*

[1] P256-M
[2] TF-M QA Reports SQUAD
[3] TF-M performance profiling tool
[4] Read the Docs

Tested platforms

The following platforms are successfully tested in this release.

- **Arm**
  - AN519
  - AN521
  - AN555
  - Corstone-1000
  - Corstone-300
  - Corstone-310
  - Musca-B1
  - Musca-S1
- **Infineon/Cypress**
  - PSoC 64
- **STM**
  - B-U585I-IOT02A
  - NUCLEO-L552ZE-Q
  - STM32H573idk
- **Nordic**
  - nRF5340
  - nRF9160
- **NuMaker-PFM**
  - M2351
  - M2354
- **NXP**
  - LPCXpresso55S69

Reference memory footprint

All measurements below are made for *AN521* platform, built TF-Mv2.0.0-RC2 on Windows 10 using Armclang v6.18 and build type MinSizeRel.

All modules are measured in bytes. Some minor modules are not shown in the table below.

Note: Profile *Medium-ARoT-less* built with disabled Firmware Update service to align with other TF-M Profiles.

Module	Base		Small		ARoT-less		Medium		Large	
	Flash	RAM	Flash	RAM	Flash	RAM	Flash	RAM	Flash	RAM
Generated (stack, stc)	112	3184	160	3184	160	3184	208	3184	272	3184
Objects	940	1064	1224	5464	1313	6152	1443	1496	1518	1496
c_w.l	190	0	690	0	690	0	690	0	930	0
platform (Secure)	5098	284	5430	284	5782	284	6154	284	6284	284
SPM	3574	165	4456	165	3946	165	6330	1353	6484	1358
sprt	274	0	1470	0	1308	0	2470	4	2454	4
MbedCrypto	N/A	N/A	25220	2108	29964	2104	29968	2104	78938	1996
PROT_attestation	N/A	N/A	2341	557	2571	1218	2571	3010	2687	3010
PROT_crypto	N/A	N/A	3866	2070	4420	16026	4420	22938	4552	25818
PROT_ITS	N/A	N/A	4830	80	4894	112	5064	1988	5068	2498
PROT_platform	N/A	N/A	N/A	N/A	478	0	520	1280	520	1280
AROT_PS	N/A	N/A	N/A	N/A	N/A	N/A	3276	4364	3276	4364
plat-form_crypto_keys	N/A	N/A	248	0	256	0	256	0	256	0
qcbor	N/A	N/A	854	0	854	0	854	0	854	0
crypto_service_p256	N/A	N/A	N/A	N/A	3534	0	3534	0	N/A	N/A
Padding	32	39	111	16	118	19	126	47	187	38
Total incl. padding	10220	4736	50900	13928	60288	29264	67884	42052	114280	45300

Known issues

Some open issues are not fixed in this release.

Descriptions	Issue links
TF-M Kconfig is broken due to build split. It will be recovered in a future release.	Not tracked

Issues fixed since v1.8.0

The following issues have been fixed since the v1.8.0 release.

Descriptions	Issue links
Arm GNU toolchain version greater than *11.2* has a linker issue in *syscall*.	https://developer.trustedfirmware.org/T1029

Reference

Copyright (c) 2023, Arm Limited. All rights reserved.

Version 1.8.1

Hotfixes

- Fix for “Partial tag comparison when using Chacha20-Poly1305 on the PSA driver API interface in CryptoCell enabled platforms” (CVE-2023-40271) (*TFMV-6*)

Copyright (c) 2023, Arm Limited. All rights reserved.

Version 1.8.0

New major features

- TF-M eRPC Test framework[1] is integrated.
- TF-M builtin key loader integration is reworked.
- Improved crypto library abstraction from TF-M Crypto service.
- *Kconfig system* is enhanced and finalized.
- Switch to upstream QCBOR[2].
- Enable PSA Authenticated Debug Access Control (ADAC)[3] support on Musca-B1 platform.
- Support Floating-Point (FP) with Arm Compiler.
- FF-M API uses signals to drive the partition scheduling instead of controlling partition context directly.
- MCUboot upgrade to v1.10.0.
- Mbed TLS upgrade to v3.4.0.
- Refine documentation restructure.
- It is optional to update copyright year in changes. Requirements of copyright note update is updated in *Contributing Process*.

New platforms supported

- *AN555.*

Tested platforms

The following platforms are successfully tested in this release.

- AN519
- AN521
- AN555
- B-U585I-IOT02A
- Corstone-1000
- LPCXpresso55S69
- Musca-B1
- Musca-S1
- nRF9160
- nRF5340
- NUCLEO-L552ZE-Q
- PSoC 64
- RSS
- STM32L562E-DK

[1] TF-M eRPC Test Framework
[2] QCBOR repository
[3] Authenticated Debug Access Control Specification

Reference memory footprint

All measurements below are made for *AN521* platform, built TF-Mv1.8.0-RC2 on Windows 10 using Armclang v6.18 and build type MinSizeRel.

All modules are measured in bytes. Some minor modules are not shown in the table below.

Note: Profile *Medium-ARoT-less* built with disabled Firmware Update service to algin with other TF-M Profiles.

Module	Base		Small		ARoT-less		Medium		Large	
	Flash	RAM	Flash	RAM	Flash	RAM	Flash	RAM	Flash	RAM
Generated (stack, stc)	112	3184	160	3184	160	3184	208	3184	272	3184
Objects	936	1064	1222	5464	1309	6152	1439	1544	1514	1544
c_w.l	190	0	568	0	808	0	808	0	808	0
platform (Secure)	5098	284	5530	284	5890	284	6256	284	6278	284
SPM	3694	1205	4560	577	4070	1205	5710	1261	5870	1266
sprt	254	0	1432	0	1270	0	2198	4	2198	4
MbedCrypto	N/A	N/A	24612	2292	46558	2296	46562	2296	80435	2408
PROT_attestation	N/A	N/A	2751	557	2981	1218	2981	3010	2981	3010
PROT_crypto	N/A	N/A	3686	2070	4240	16026	4240	22938	4368	24986
PROT_ITS	N/A	N/A	4826	80	4882	112	5056	1988	5068	2468
PROT_platform	N/A	N/A	N/A	N/A	480	0	522	1280	522	1280
AROT_PS	N/A	N/A	N/A	N/A	N/A	N/A	3340	6444	3340	6444
Padding	40	23	119	28	158	35	142	39	180	66
Total incl. padding	10324	5760	50576	14536	73916	30512	80572	44272	114924	46944

Known issues

Some open issues are not fixed in this release.

Descriptions	Issue links
Arm GNU toolchain version greater than *11.2* has a linker issue in *syscall*.	https://developer.trustedfirmware.org/T1029
In SFN model, input data size value in structure *psa_msg_t* passed to RoT Services might decrease after *psa_read()* or *psa_skip()*.	https://developer.trustedfirmware.org/T1030

Issues fixed since v1.7.0

The following issues have been fixed since v1.7.0 release.

Descriptions	Issue links
PSoC 64 platform builds and works well with ArmClang v6.13 and later.	Not tracked
LPCXpresso55S690 platform works well in profile small only.	Not tracked

Issues closed since v1.7.0

The following issues in v1.7.0 known issue list are closed. These issues are related to platform specific implementations or 3rd-party tools. Therefore they won't be fixed by TF-M.

Descriptions	Issue links
Latest CMake (3.25) has an issue with PSA Arch test	https://github.com/ARM-software/psa-arch-tests/issues/335

Reference

Copyright (c) 2023, Arm Limited. All rights reserved.

Release Cadence and Process

The project aims to do a release once every 9 months. The planned dates are listed in the *Future release plans*

The releases are to be performed and tagged on a dedicated release branch. The release process is initiated by an announcement in TF-M mailing list and followed by the creation of a release branch. Normal development on the main branch is not blocked and can be continued. The testing will be performed on release candidates and depending on issues found, additional candidates may be created to fix and retest the issues.

The Platform owners are expected to verify their platforms and confirm the correct operations or provide fixes in a timely manner to include in the release. The release notes will list all verified platforms. The platforms in Trusted Firmware OpenCI are automatically tested and any issues found shall be fixed.

After the final tag, the changes from the release branch will be back ported to the main branch.

```
@startuml
hide time-axis

concise "main branch" as main
concise "release branch v1.1.x" as rel1
concise "release branch v1.2.x" as rel2

@main
    -3 is development
    @0 <-> @8 : release cadence: ~9 months

@rel1
    0 is rc1
    main -> rel1 : start
    +1 is rc2
    +1 is v1.1.0
    +1 is {-}
    rel1 -> main : back port
    +1 is v1.1.1 #pink
    +1 is {-}
    rel1 -> main : cherry-pick
    +3 is END #white
    +1 is {hidden}

@rel2
    8 is rc1
    main -> rel2 : start
    +1 is v1.2.0
    +1 is {-}

    rel2 -> main : back port
```

(continues on next page)

(continued from previous page)

```
    @0 <-> @3 : release process
    @4 <-> @5 : hotfix

caption Release process

@enduml
```

Although this document specifies the release cadence, this does not preclude an adhoc release for specific project requirements.

Note: When a new release starts the previous release branch obsoletes and is a subject for removal as shown for *v1.1.x* on the diagram above. At any moment only the latest release branch is maintained. The release tags will point to a commit in detached head state.

Release Version Scheme

Trusted Firmware-M uses a semantic versioning scheme. A version number is compiled as a dot separated set of numbers:

TF-Mv<MAJOR>.<MINOR>.<HOTFIX>

- <MAJOR>: Major release version for significant feature and API changes.
- <MINOR>: Minor release version for incremental features and API changes.
- <HOTFIX>: Used only for backporting **critical bug fix/security patches**.

Long Term Support (LTS)

From v2.1.0, TF-M project will provide LTS branches. Every alternate release will be an LTS release maintained for 3 years. The LTS release will be synchronized with Mbed TLS project to be aligned with LTS releases cadence. Mbed TLS is used as the default cryptography library by the TF-M Crypto service.

The main purpose of TF-M LTS is to offer a maintained and PSA certified TF-M codebase for the whole period of LTS. Without LTS, every TF-M based product has to recall or update PSA certification with changes (especially security fixes) to the codebase. Some changes like fixes for critical bugs or security vulnerabilities are essential to keep a product secure with valid PSA certificates. The recertification is a time and resource-consuming process, putting extra burden on the platform owners. Moreover, this procedure is inefficient and unscalable to be done on a huge number of PSA Certified TF-M platforms.

TF-M intends to centralise PSA certification of the common code and carry it on for an LTS lifetime, ensuring that code is free from known bugs and security vulnerabilities. For this, each LTS branch will be initially PSA certified for a selected reference platform and recertified again on every TF-M release. Please see the process<link> below for the details. All platforms, based on LTS version can hold PSA certification obtained once without a need for recertification on updates if no vulnerability is found in a platform specific code.

LTS content

On every TF-M release the following items are backported from the **release** branch to active LTS branches:

- Critical bugs
- Security vulnerability fixes
- A new platform port with code changes restricted to the relevant platform folder only.

A full release cycle is performed on each updated LTS branch. If no items above were delivered to the main branch since the last release, then LTS branches remain unchanged with valid PSA certificate for that version.

Frequency and duration

- A new LTS branch is created on every other TF-M release i.e. every 18 months.
- Each LTS branch is maintained for 3 years.

```
@startuml
hide time-axis

concise "main branch" as main
concise "release branch v2.1.x (LTS)" as lts21
concise "release branch v2.2.x" as rel22
concise "release branch v2.3.x (LTS)" as lts23
concise "release branch v2.4.x" as rel24
concise "release branch v2.5.x (LTS)" as lts25

@main
    -1 is development
    @0 <-> @6 : release cadence: ~9 months

@lts21
    0 is v2.1.0
    main -> lts21 : start
    +1 is "PSA cert" #aquamarine
    +2 is "v2.1.0-LTS"
    +2 is {-}
    +19 is END #white

@rel22
    6 is v2.2.0
    main -> rel22 : start
    +1 is {-}
    +1 is v2.2.1 #pink
    +1 is {-}
    rel22 -> main : back port
    rel22 -> lts21 : back port
    +3 is END #white
    +1 is {hidden}

@lts23
    12 is v2.3.0
    main -> lts23 : start
    +1 is "PSA cert" #aquamarine
    +2 is "v2.3.0-LTS"
    +2 is {-}

    @8 <-> @9 : hotfix

@rel24
    18 is v2.4.0
    main -> rel24 : start
    +1 is {-}
    +2 is v2.4.1 #pink
    +1 is {-}
    rel24 -> main : back port
    rel24 -> lts21 : back port
    rel24 -> lts23 : back port
    +2 is END #white
```

(continues on next page)

(continued from previous page)

```
    @0 <-> @12 : LTS release every 18 months

@lts25
    24 is v2.5.0
    main -> lts25 : start

    @0 <-> @24 : LTS lifetime: 3 years

caption Long Time Support (LTS) process

@enduml
```

LTS usage scenario

- Once released, the code (ex:*TF-MvX.Y.Z*) is submitted for PSA certification using the reference platform *Musca-B1 Platform Specifics*.
- After obtaining PSA certification the release branch is tagged *TF-MvX.Y.Z-LTS* creating a base for PSA certification of downstream projects.
- Ad hoc security fixes on the current **release** branch resulting in a new TF-M release will be backported to LTS branches under maintenance with consequent minor releases on them. Each LTS minor release may be a subject for PSA recertification with evaluation of the code changes (delta certification).
- Platform independent TF-M fixes are evaluated once & available to PSA Certified platforms on the new LTS release for rebase without the need for individual recertification.
- Bug fixes (other than security fixes) are backported from the main branch to active LTS branches and made part of the next regular release. The LTS release will be subject to PSA Certification.

Q&A

1. **What if the release (e.g. TF-MvX.Y.Z) fails PSA certification?**

   A failure on PSA certification means a critical bug or vulnerability and requires code modification in the form of a hot fix followed by a new minor release.

2. **What is time gap between release and LTS tag?**

   It depends on PSA Lab Certification Body but is expected to be about 1 month.

3. **What if a security vulnerability is found in platform-specific code?**

   A vulnerable platform shall provide a fix for it and perform PSA certification independently.

Copyright (c) 2020-2024, Arm Limited. All rights reserved.

Version	Date	PSA-arch tag/hash
v2.1.4	2025-11-21	v23.06_API1.5_ADAC_EAC
v2.1.3	2025-08-01	v23.06_API1.5_ADAC_EAC
v2.1.2	2025-04-14	v23.06_API1.5_ADAC_EAC
v2.1.1	2024-11-01	v23.06_API1.5_ADAC_EAC
v2.1.0	2024-05-13	v23.06_API1.5_ADAC_EAC
v2.0.0	2023-11-28	v23.06_API1.5_ADAC_EAC
v1.8.1	2023-09-08	cf8bd71
v1.8.0	2023-04-28	cf8bd71
v1.7.0	2022-12-08	cf8bd71
v1.6.1	2022-11-16	f7e8495
v1.6.0	2022-04-28	f7e8495
v1.5.0	2021-11-30	v21.10_API1.3_ADAC_ALPHA-1
v1.4.0	2021-08-03	51ff2bd
v1.3.0	2021-04-07	b0635d9
v1.2.0	2020-11-25	90c8e680
v1.1	2020-07-15	1f960947
v1.0	2020-03-27	v20.03_API1.0

Future release plans

The dates below are tentative and subject to change.

Version	Feature Freeze	Release
v2.3.0-LTS	2026-APR (TBC)	2026-APR (TBC)
v2.4.0	2026-Q4 (TBC)	2026-Q4 (TBC)

Please refer to *Release Cadence and Process* for interpreting version numbers.

SPDX-License-Identifier: BSD-3-Clause

SPDX-FileCopyrightText: Copyright The TrustedFirmware-M Contributors

Roadmap

TF-M has been under active development since it was launched in Q1'18. It is being designed to include

1. Secure boot ensuring integrity of runtime images and responsible for firmware upgrade.
2. Runtime firmware consisting of TF-M Core responsible for secure isolation, execution and communication aspects. and a set of Secure Services providing services to the Non-Secure and Secure Applications. The secures services currently supported are Secure Storage, Cryptography, Firmware Update, Attestation and Platform Services

If you are interested in collaborating on any of the roadmap features or other features, please mail TF-M mailing list

Supported Features

- PSA Firmware Framework v1.0, 1.1 Extension including IPC and SFN modes.
- PSA Level1, 2 and 3 Isolation.
- Secure Boot (mcuboot upstream) including generic fault injection mitigations
- PSA Protected Storage, Internal Trusted Storage v1.0 and Encrypted ITS

- PSA Cryptov1.0 (uses Mbed TLS v3.4.0)
- PSA Initial Attestation Service v1.0
- PSA Firmware Update v1.0
- PSA ADAC Specification Implementation
- Base Config
- kconfig based configuration
- Profile Small, Medium, ARoT-less Medium, Large
- Secure Partition Interrupt Handling, Pre-emption of SPE execution
- Platform Reset Service
- Dual CPU
- Open Continuous Integration (CI) System
- Boot and Runtime Crypto Hardware Integration
- Fault Injection Handling library to mitigate against physical attacks
- Threat Model
- Arm v8.1-M Privileged Execute Never (PXN) attribute and Thread reentrancy disabled (TRD)
- FPU, MVE Support
- CC-312 PSA Cryptoprocessor Driver Interface
- Secure Storage - Key Diversification Enhancements
- Build System - Separate Secure and Non-Secure builds

CQ1'24

- Supporting multiple clients (Hybrid Platforms) i.e. TF-M supporting multiple on core and off core clients on heterogeneous (e.g. Cortex-A + Cortex-M platforms) Mailbox API etc.
- PSA Crypto layer for mcuboot/BL2
- Enable PSA Crypto Client from Non-Secure via. IPC
- Long Term Stable (LTS) Release preparations

Future

- TF-M v2.1.0 Long Term Stable (LTS) Release
- Demonstrating TLS in Non-Secure using PSA Crypto APIs in TF-M
- Implement support for multiple clients (Hybrid Platforms)
- Build System Enhancements - Separate Secure, Non-Secure Builds
- Remote Test Infrastructure
- MISRA testing/documentation
- TF-M Performance - Further Benchmarking and Optimization
- Scheduler - Multiple Secure Context Implementation
- Arm v8.1-M Architecture Enablement - PAC/BTI
- PSA FWU Service Enhancements
- PSA ADAC Spec - Enhancements and Testing

- Arm v8.1-M Unprevileged Debug
- [Secure Storage] Extended PSA APIs
- [Audit Logs] Secure Storage, Policy Manager
- PSA FF Lifecycle API
- Fuzz Testing

Copyright (c) 2017-2024, Arm Limited. All rights reserved.

Glossary of terms and abbreviations

AAPCS
ARM Architecture Procedure Call Standard: The AAPCS defines how subroutines can be separately written, separately compiled, and separately assembled to work together. It describes a contract between a calling routine and a called routine

Application RoT
PSA term. The security domain in which additional security services are implemented. Also referred as ARoT.

HAL
Hardware Abstraction Layer: Interface to abstract hardware-oriented operations and provides a set of APIs to the upper layers.

ITS
Internal Trusted Storage
One of PSA services provided by TF-M.

MPC
Memory Protection Controller: Bus slave-side security controller for memory regions.

MPU
Memory Protection Unit: Hardware component providing privilege control.

NSPE
Non Secure Processing Enviroment: PSA term. In TF-M this means non secure domain typically running an OS using services provided by TF-M.

PPC
Peripheral Protection Controller: Bus slave-side security controller for peripheral access.

PS
Protected Storage
One of PSA services provided by TF-M.

PSA
PSA term. Platform Security Architecture.

PSA RoT
PSA term. This defines the most trusted security domain within a PSA system. Also referred as PRoT.

PSA-FF
PSA term. Platform Security Architecture Firmware Framework.

PSA-FF-M
PSA term. Platform Security Architecture Firmware Framework for M.

RoT
Root of Trust: PSA term. This is the minimal set of software, hardware and data that is implicitly trusted in the platform — there is no software or hardware at a deeper level that can verify that the Root of Trust is authentic and unmodified.

RoT Service
PSA term. A set of related security operations that are implemented in a Secure Partition.

S/NS
Secure/Non-secure: The separation provided by TrustZone hardware components in the system.

SAU
Secure Attribution Unit: Hardware component providing isolation between Secure, Non-secure Callable and Non-secure addresses.

SFN
Secure Function: The function entry to a secure service. Multiple SFN per SS are permitted.

SP
Secure Partition
A logical container for secure services.

SPE
Secure Processing Environment: PSA term. In TF-M this means the secure domain protected by TF-M.

SPM
Secure Partition Manager
The TF-M component responsible for enumeration, management and isolation of multiple Secure Partitions within the TEE.

SPRT
Secure Partition Runtime: The TF-M component responsible for Secure Partition runtime functionalities.

SPRTL
Secure Partition Runtime Library: A library contains the SPRT code and data.

SS
Secure Service: A component within the TEE that is atomic from a security/trust point of view, i.e. which is viewed as a single entity from a TF-M point of view.

SVC
SuperVisor Call: ARMv7M assembly instruction to call a privileged handler function

TBSA-M
Trusted Base System Architecture for M. TBSA term. See Trusted Base System Architecture for M

TFM
TF-M
Trusted Firmware-M or Trusted Firmware for M-class. ARM TF-M provides a reference implementation of secure world software for ARMv8-M.

Reference

Firmware Framework for M (FF-M)

Trusted Base System Architecture for M

Copyright (c) 2017-2024, Arm Limited. All rights reserved.

TF-M Platforms

Analog Devices, Inc. Platforms

MAX32657

Introduction

The MAX32657 microcontroller (MCU) is an advanced system-on-chip (SoC) featuring an Arm® Cortex®-M33 core with single-precision floating point unit (FPU) with digital signal processing (DSP) instructions, large flash and SRAM memories, and the latest generation Bluetooth® 5.4 Low Energy (LE) radio. The nano-power modes increase battery life substantially.

MAX32657 1MB flash and 256KB RAM split to define section for MCUBoot, TF-M (S), Zephyr (NS) and storage that used for secure services and configurations. Default layout of MAX32657 is listed in below table.

Name	Address[Size]	Comment
boot	0x1000000[64K]	MCU Bootloader
slot0	0x1010000[320k]	Secure image slot0 (TF-M)
slot0_ns	0x1060000[576k]	Non-secure image slot0
slot1	0x10F0000[0k]	Updates slot0 image
slot1_ns	0x10F0000[0k]	Updates slot0_ns image
storage	0x10f0000[64k]	File system, persistent storage

RAM	Address[Size]	Comment
secure_ram	0x30000000[64k]	Secure memory
non_secure_ram	0x20010000[192k]	Non-Secure memory

Secure Boot ROM

MAX32657 has Secure Boot ROM that used to authenticate user code via ECDSA 256 public key. The Secure Boot ROM is disabled on default, to enable it user need to provision device first.

ADI provides enable_secure_boot.py (under <CMAKE_BINARY_DIR>/lib/ext/tesa-toolkit-src/devices/max32657/scripts/bl1_provision) script to simply provision the device. This script reads user certificate via command line parameter then writes user key on the device and disables debug interface.

To create pub & private key pair for MAX32657 run:

```
openssl ecparam -out <MY_CERT_FILE.pem> -genkey -name prime256v1
```

Note: Debug interface will be disabled after secure boot is enabled. User must write final firmware before provisioning the device. It can be written during device provision, Just add your final firmware hex file in JLinkScript under <CMAKE_BINARY_DIR>/lib/ext/tesa-toolkit-src/devices/max32657/scripts/bl1_provision folder.

After secure boot has been enabled BL2 image must be signed with user certificate otherwise Secure Boot ROM will not validate BL2 image and will not execute it. The sign process will be done automatically if BL1 be ON -DBL1=ON The sign key can be sepecified over command line option -DTFM_BL2_SIGNING_KEY_PATH=<MY_KEY_FILE> or by setting the flag in <TF-M base folder>/platform/ext/target/adi/max32657/config.cmake Development purpose test certificate is here: <CMAKE_BINARY_DIR>/lib/ext/tesa-toolkit-src/devices/max32657/keys/bl1_dummy.pem It shall not been used for production purpose just for development purpose.

Note: The signature generation depends on ecdsa that's have to be installed:

```
pip3 install ecdsa
```

Building TF-M

This platform port supports TF-M regression tests (Secure and Non-secure) with Isolation Level 1.

To build S and NS application, run the following commands:

Note: Only GNU toolchain is supported.

Note: Only "profile_small" predefined profile is supported.

Prepare the tf-m-tests repository inside the TF-M base folder.

```
cd <TF-M base folder>
git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
```

```
cd <TF-M base folder>/tf-m-test/tests_reg

cmake -S spe -B build_spe \
      -G"Unix Makefiles"              \
      -DTFM_PLATFORM=adi/max32657     \
      -DCONFIG_TFM_SOURCE_PATH=<TF-M base folder>/trusted-firmware-m \
      -DTFM_TOOLCHAIN_FILE=<TF-M base folder>/trusted-firmware-m/toolchain_GNUARM.cmake \
      -DTEST_S=OFF                    \
      -DTEST_NS=ON                    \
      -DTFM_NS_REG_TEST=ON            \
      -DMCUBOOT_LOG_LEVEL="INFO"      \
      -DTFM_ISOLATION_LEVEL=1
cmake --build build_spe -- install

cmake -S . -B build_test     \
      -G"Unix Makefiles"     \
      -DCONFIG_SPE_PATH=<TF-M base folder>/tf-m-tests/tests_reg/build_spe/api_ns \
      -DTFM_TOOLCHAIN_FILE=cmake/toolchain_ns_GNUARM.cmake \
      -DTFM_NS_REG_TEST=ON
cmake --build build_test
```

Merge and Flash Images

Follow the steps below to program the flash with a compiled TF-M image (i.e. S, NS or both).

Generate Intel hex files from the output binary (bin) files as follows:

```
srec_cat build_test/bin/tfm_ns_signed.bin -binary --offset 0x01060000 -o build_test/bin/tfm_ns_signed.
↪hex -intel
```

Merge hex files as follows:

```
srec_cat.exe build_spe/bin/bl2.hex -Intel build_spe/bin/tfm_s_signed.hex -Intel build_test/bin/tfm_ns_
↪signed.hex -Intel -o tfm_merged.hex -Intel
```

Note: Use bl2_signed.hex instead bl2.hex if Secure Boot ROM is enabled.

Flash them with JLink as follows:

```
JLinkExe -device MAX32657 -if swd -speed 2000 -autoconnect 1
J-Link>h
J-Link>r
J-Link>erase
J-Link>loadfile build_spe/bin/tfm_merged.hex
```

BL2 and TF-M Provisioning

On default -DPLATFORM_DEFAULT_PROVISIONING=ON and -DTFM_DUMMY_PROVISIONING=ON which will use default provisioning and dummpy keys, these configuration is fine for development purpose but for production customer specific keys shall be used Provisioning bundles can be generated with the -DPLATFORM_DEFAULT_PROVISIONING=OFF flag. The provisioning bundle binary will be generated and it's going to contain the provisioning code and provisioning values.

If -DPLATFORM_DEFAULT_PROVISIONING=OFF and -DTFM_DUMMY_PROVISIONING=ON then the keys in the tf-m/platform/ext/target/common/provisioning/provisioning_config.cmake and the default MCUBoot signing keys will be used for provisioning.

If -DPLATFORM_DEFAULT_PROVISIONING=OFF and -DTFM_DUMMY_PROVISIONING=OFF are set then unique values can be used for provisioning. The keys and seeds can be changed by passing the new values to the build command, or by setting the -DPROVISIONING_KEYS_CONFIG flag to a .cmake file that contains the keys. An example config cmake file can be seen at tf-m/platform/ext/target/common/provisioning/provisioning_config.cmake. Otherwise new random values are going to be generated and used. For the image signing the ${MCUBOOT_KEY_S} and ${MCUBOOT_KEY_NS} will be used. These variables should point to .pem files that contain the code signing private keys. The public keys are going to be generated from these private keys and will be used for provisioning. The hash of the public key is going to be written into the provisioning_data.c automatically.

If -DMCUBOOT_GENERATE_SIGNING_KEYPAIR=ON is set then a new mcuboot signing public and private keypair is going to be generated and it's going to be used to sign the S and NS binaries.

The new generated keypair can be found in the <build dir>/bin folder or in the <install directory>/image_signing/keys after installation. The generated provisioning_data.c file can be found at <build directory>/platform/target/provisioning/provisioning_data.c

---

Note: The provisioning bundle generation depends on pyelftools that's have to be installed:

```
pip3 install pyelftools
```

UART Console

MAX32657 has one UART (UART0) peripheral which is routed for Non-Secure console output by default. S and NS firmware can not use UART at the same time. If TFM_S_REG_TEST been defined the UART console will be routed to the Secure side otherwise it will be on NS side.

---

*Copyright 2025 Analog Devices, Inc. All rights reserved. *SPDX-License-Identifier: BSD-3-Clause*

Copyright (c) 2025, Analog Devices, Inc. All rights reserved.

Arm platforms

Corstone-1000

Introduction

The ARM's Corstone-1000 platform is a reference implementation of PSA FF-M architecture where NSPE and SPE environments are partitioned/isolated into Cortex-A35 and Cortex-M0+ respectively.

Cortex-M0+ acting as Secure Enclave is the Root-of-trust of SoC. Its software comprises of two boot loading stages, i.e. Bl1 and Bl2 (based on mcuboot) and TF-M as run time software. Cortex-A35, also referred as host, is treated as non-secure from the Secure Enclave perspective. The Cortex-A35 is brought out of rest by Secure Enclave in aarch64 bit mode, and boots the software ecosystem based on linux, u-boot, UEFI run time services, TF-A, Secure Partitions and Optee.

The communication between NSPE and SPE is based on PSA IPC protocol running on top of the RSE communication protocol. The Corstone-1000 supports only the *Embed protocol*, and the ATU support is removed.

The secure enclave subsystem has ARM's CC-312 (Crypto Cell) hardware to accelerate cryptographic operations. Additionaly, platform supports Secure Debug using SDC-600 as the communication interface between host debugger and platform target. The platform has the build option to enable secure debug protocol to unlock debug ports during boot time. The protocol is based on ARM's ADAC (Authenticated Debug Access Control) standard.

System boot

- The SoC reset brings Secure Enclave (SE), that is Cortex-M0+, out of rest.
- SE executes the BL1 ROM code based on mcuboot.
- BL1 load, verifies and transfer execution to BL2 which is again based on mcuboot.
- BL2 loads and verifies TF-M and host's initial boot loader image.
- BL2 transfer the execution to the TF-M.
- During TF-M initialization, the host is taken out of rest.
- Hashes of the keys used for image verification are stored in the OTP memory.

Build

Platform solution

The platform binaries are build using Yocto. Below is the user guide:

Arm Corstone-1000 User Guide

Secure Test

This section can be used to test the secure enclave software independently from the host. The below configuration builds the secure enclave binaries with CI test frame integrated. On boot, secure enclave softwares stack is brought up, and CI tests starts executing at the end of the initialization process. In the below configuration, host software support is disabled, and meant only to test/verify the secure enclave softwares.

FVP

- Download Corstone-1000 FVP from : Arm Ecosystem FVPs
- Install FVP by running the shell script.
- Running of the binary will boot secure enclave software stack and at the end all CI test from tf-m-test along with platform specific tests are executed.

```
cd <tf-m-root>/
cmake -B build/ -S . -DCMAKE_BUILD_TYPE=Debug -DTFM_TOOLCHAIN_FILE=<tf-m-root>/
↪toolchain_GNUARM.cmake -DTFM_PLATFORM=arm/corstone1000 -DPLATFORM_IS_FVP=TRUE -
↪DTEST_NS=OFF -DTEST_S=ON -DEXTRA_S_TEST_SUITE_PATH=platform/ext/target/arm/
↪corstone1000/ci_regression_tests/
cmake --build build -- install
dd conv=notrunc bs=1 if=build/install/outputs/bl1_1.bin of=build/install/outputs/bl1.bin seek=0
dd conv=notrunc bs=1 if=build/install/outputs/bl1_provisioning_bundle.bin of=build/install/outputs/bl1.
↪bin seek=40960
./platform/ext/target/arm/corstone1000/create-flash-image.sh build/install/outputs cs1000.bin

<path-to-FVP-installation>/models/Linux64_GCC-9.3/FVP_Corstone-1000 -C board.flashloader0.fname=
↪"none" -C se.trustedBootROMloader.fname="build/install/outputs/bl1.bin" -C board.xnvm_size=64 -C se.
↪trustedSRAM_config=6 -C se.BootROM_config="3" -C board.smsc_91c111.enabled=0  -C board.
↪hostbridge.userNetworking=true --data board.flash0=build/install/outputs/cs1000.bin@0x68000000 -C␣
↪diagnostics=4 -C disable_visualisation=true -C board.se_flash_size=8192 -C diagnostics=4  -C disable_
↪visualisation=true
```

FPGA

- Follow the above pointed platform user guide to setup the FPGA board.
- Use the BL1 generated from the below commands to place it inside FPGA board SD Card.
- Use the cs1000.bin created from the below commands to place it inside FPGA board SD Card.

```
cd <tf-m-root>/
cmake -B build/ -S . -DCMAKE_BUILD_TYPE=Debug -DTFM_TOOLCHAIN_FILE=<tf-m-root>/
↪toolchain_GNUARM.cmake -DTFM_PLATFORM=arm/corstone1000 -DTEST_NS=OFF -DTEST_
↪S=ON -DTEST_S_PS=OFF -DTEST_S_PLATFORM=OFF -DEXTRA_S_TEST_SUITE_
↪PATH=platform/ext/target/arm/corstone1000/ci_regression_tests/
cmake --build build -- install
dd conv=notrunc bs=1 if=build/install/outputs/bl1_1.bin of=build/install/outputs/bl1.bin seek=0
dd conv=notrunc bs=1 if=build/install/outputs/bl1_provisioning_bundle.bin of=build/install/outputs/bl1.
↪bin seek=40960
./platform/ext/target/arm/corstone1000/create-flash-image.sh build/install/outputs cs1000.bin
cp build/install/outputs/bl1.bin <path-to-FPGA-SD-CARD>/SOFTWARE/
cp build/install/outputs/cs1000.bin <path-to-FPGA-SD-CARD>/SOFTWARE/
```

FPGA build can not compile all the CI tests into a single build as it exceeds the available RAM size. So there is a need to select few tests but not all. The above configuration disable build of -DTEST_S_PS and -DTEST_S_PLATFORM. Other test configurations are:

- -DTEST_S_ATTESTATION=ON/OFF
- -DTEST_S_CRYPTO=ON/OFF
- -DTEST_S_ITS=ON/OFF
- -DTEST_S_PS=ON/OFF
- -DTEST_S_PLATFORM=ON/OFF

Copyright (c) 2021-2024, Arm Limited. All rights reserved.

Corstone SSE-300 with Ethos-U55/U65 Example Subsystem for Arm Ecosystem FVP and for MPS3 (AN547, AN552)

Introduction

Corstone-300 is an Arm reference subsystem for secure System on Chips containing an Armv8.1-M Cortex-M55 processor and an Ethos-U55/U65 neural network processor. It is an MPS3 based platform with the usual MPS3 peripherals.

This platform port supports all TF-M regression tests (Secure and Non-secure) with Isolation Level 1 and 2.

Note: For Ethos-U55/U65 IP this platform support only provides base address, interrupt number and an example NPU setup as non-secure, unprivileged.

Note: For Armclang compiler v6.18 or later version is required.

Building TF-M

Follow the instructions in Building instructions.

Build instructions with platform name: arm/mps3/corstone300/an547

-DTFM_PLATFORM=arm/mps3/corstone300/an547

Note: For Ethos-U55/U65 IP this platform support only provides base address, interrupt number and an example NPU setup as non-secure, unprivileged.

Build instructions with platform name: arm/mps3/corstone300/an552

-DTFM_PLATFORM=arm/mps3/corstone300/an552

Note: For Ethos-U55/U65 IP this platform support only provides base address, interrupt number and an example NPU setup as non-secure, unprivileged.

Build instructions with platform name: arm/mps3/corstone300/fvp

-DTFM_PLATFORM=arm/mps3/corstone300/fvp

Note: For Ethos-U55/U65 IP this platform support only provides base address, interrupt number and an example NPU setup as non-secure, unprivileged.

Note: The built binaries can be run on the Corstone-300 Ethos-U55/U65 Ecosystem FVP (FVP_SSE300_MPS3). At least Ecosystem FVP version 11.22 is required.

Note: Provisioning bundles can be generated with the -DPLATFORM_DEFAULT_PROVISIONING=OFF flag. The provisioning bundle binary will be generated and it's going to contain the provisioning code and provisioning values.

Note: If -DPLATFORM_DEFAULT_PROVISIONING=OFF and -DTFM_DUMMY_PROVISIONING=ON then the keys in the tf-m/platform/ext/target/arm/mps3/common/provisioning/provisioning_config.cmake and the default MCUBoot signing keys will be used for provisioning.

If -DPLATFORM_DEFAULT_PROVISIONING=OFF and -DTFM_DUMMY_PROVISIONING=OFF are set then unique values can be used for provisioning. The keys and seeds can

be changed by passing the new values to the build command, or by setting the -DPROVISIONING_KEYS_CONFIG flag to a .cmake file that contains the keys. An example config cmake file can be seen at tf-m/platform/ext/target/arm/mps3/common/provisioning/provisioning_config.cmake. Otherwise new random values are going to be generated and used. For the image signing the ${MCUBOOT_KEY_S} and ${MCUBOOT_KEY_NS} will be used. These variables should point to .pem files that contain the code signing private keys. The public keys are going to be generated from these private keys and will be used for provisioning. The hash of the public key is going to be written into the provisioning_data.c automatically.

If -DMCUBOOT_GENERATE_SIGNING_KEYPAIR=ON is set then a new mcuboot signing public and private keypair is going to be generated and it's going to be used to sign the S and NS binaries.

The new generated keypair can be found in the <build dir>/bin folder or in the <install directory>/image_signing/keys after installation. The generated provisioning_data.c file can be found at <build directory>/platform/target/provisioning/provisioning_data.c

---

Note: The provisioning bundle generation depends on pyelftools that's have to be installed:

```
pip3 install pyelftools
```

To run the example code on Corstone SSE-300 with Ethos-U55/U65 Example Subsystem for MPS3 (AN547)

FPGA image is available to download here

If the link above is not working just go to Arm PDH and search for AN547.

To run BL2 bootloader, TF-M example application and tests in the MPS3 board, it is required to have AN547 image in the MPS3 board SD card. The image should be located in <MPS3 device name>/MB/HBI<BoardNumberBoardrevision>/AN547

The MPS3 board tested is HBI0309C.

1. Copy bl2.bin and tfm_s_ns_signed.bin files from build dir to <MPS3 device name>/SOFTWARE/
2. Rename tfm_s_ns_signed.bin to tfm.bin (Filename should not be longer than 8 charachters.)
3. Open <MPS3 device name>/MB/HBI0309C/AN547/images.txt
4. Update the images.txt file as follows:

```
TITLE: Arm MPS3 FPGA prototyping board Images Configuration File

[IMAGES]
TOTALIMAGES: 2                  ;Number of Images (Max: 32)

IMAGE0UPDATE: AUTO              ;Image Update:NONE/AUTO/FORCE
IMAGE0ADDRESS: 0x00000000       ;Please select the required executable program
IMAGE0FILE: \SOFTWARE\bl2.bin
IMAGE1UPDATE: FORCEQSPI
IMAGE1ADDRESS: 0x00000000
IMAGE1FILE: \SOFTWARE\tfm.bin
```

5. Close <MPS3 device name>/MB/HBI0309C/AN547/images.txt
6. Unmount/eject the <MPS3 device name> unit
7. Reset the board to execute the TF-M example application
8. After completing the procedure you should be able to visualize on the serial port (baud 115200 8n1) the following messages:

```
[INF] Starting bootloader
[INF] Beginning BL2 provisioning
[WRN] TFM_DUMMY_PROVISIONING is not suitable for production! This device is NOT SECURE
[INF] Image index: 1, Swap type: none
[INF] Image index: 0, Swap type: none
[INF] Bootloader chainload address offset: 0x0
[INF] Jumping to the first image slot
[INF] Beginning TF-M provisioning
[WRN] TFM_DUMMY_PROVISIONING is not suitable for production! This device is NOT SECURE
[WRN] This device was provisioned with dummy keys. This device is NOT SECURE
[Sec Thread] Secure image initializing!
TF-M isolation level is: 0x00000001
Booting TF-M v1.8.1
```

Note: Some of the messages above are only visible when CMAKE_BUILD_TYPE is set to Debug.

Note: If -DPLATFORM_DEFAULT_PROVISIONING=OFF is set then the provisioning bundle has to be placed on the 0x10022400 address by copying provisioning_bundle.bin and renaming it to prv.bin, then extending the images.txt with:

```
IMAGE2UPDATE: AUTO
IMAGE2ADDRESS: 0x00022400
IMAGE2FILE: \SOFTWARE\prv.bin
```

To run the example code on Corstone SSE-300 with Ethos-U55/U65 Example Subsystem for MPS3 (AN552)

FPGA image is available to download here

If the link above is not working just go to Arm PDH and search for AN552.

To run BL2 bootloader and TF-M example application and tests in the MPS3 board, it is required to have AN552 image in the MPS3 board SD card. The image should be located in <MPS3 device name>/MB/HBI<BoardNumberBoardrevision>/AN552

The MPS3 board tested is HBI0309C.

1. Copy bl2.bin and tfm_s_ns_signed.bin files from build dir to <MPS3 device name>/SOFTWARE/
2. Rename tfm_s_ns_signed.bin to tfm.bin (Filename should not be longer than 8 charachters.)
3. Open <MPS3 device name>/MB/HBI0309C/AN552/images.txt
4. Update the images.txt file as follows:

```
TITLE: Arm MPS3 FPGA prototyping board Images Configuration File

[IMAGES]
TOTALIMAGES: 2                  ;Number of Images (Max: 32)

IMAGE0UPDATE: AUTO              ;Image Update:NONE/AUTO/FORCE
IMAGE0ADDRESS: 0x00000000       ;Please select the required executable program
IMAGE0FILE: \SOFTWARE\bl2.bin
IMAGE1UPDATE: FORCEQSPI
IMAGE1ADDRESS: 0x00000000
IMAGE1FILE: \SOFTWARE\tfm.bin
```

5. Close <MPS3 device name>/MB/HBI0309C/AN552/images.txt
6. Unmount/eject the <MPS3 device name> unit

7. Reset the board to execute the TF-M example application
8. After completing the procedure you should be able to visualize on the serial port (baud 115200 8n1) the following messages:

```
[INF] Starting bootloader
[INF] Beginning BL2 provisioning
[WRN] TFM_DUMMY_PROVISIONING is not suitable for production! This device is NOT SECURE
[INF] Image index: 1, Swap type: none
[INF] Image index: 0, Swap type: none
[INF] Bootloader chainload address offset: 0x0
[INF] Jumping to the first image slot
[INF] Beginning TF-M provisioning
[WRN] TFM_DUMMY_PROVISIONING is not suitable for production! This device is NOT SECURE
[WRN] This device was provisioned with dummy keys. This device is NOT SECURE
[Sec Thread] Secure image initializing!
TF-M isolation level is: 0x00000001
Booting TF-M v1.8.1
```

Note: Some of the messages above are only visible when CMAKE_BUILD_TYPE is set to Debug.

Note: If -DPLATFORM_DEFAULT_PROVISIONING=OFF is set then the provisioning bundle has to be placed on the 0x10022400 address by copying provisioning_bundle.bin and renaming it to prv.bin, then extending the images.txt with:

```
IMAGE2UPDATE: AUTO
IMAGE2ADDRESS: 0x00022400
IMAGE2FILE: \SOFTWARE\prv.bin
```

To run the example code on Corstone-300 Ethos-U55/U65 Ecosystem FVP

FVP is available to download here

1. Install the FVP
2. Copy bl2.axf and tfm_s_ns_signed.bin files from build dir to <FVP installation path>/models/Linux64_GCC-9.3/
3. Navigate to the same directory and execute the following command to start FVP:

```
$ ./FVP_Corstone_SSE-300_Ethos-U55 -a cpu0*="bl2.axf" --data "tfm_s_ns_signed.bin
↪"@0x38000000
```

4. After completing the procedure you should be able to see similar messages to this on the serial port (baud 115200 8n1):

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[INF] Starting bootloader
[INF] Beginning BL2 provisioning
[WRN] TFM_DUMMY_PROVISIONING is not suitable for production! This device is NOT SECURE
[INF] Image index: 1, Swap type: none
[INF] Image index: 0, Swap type: none
[INF] Bootloader chainload address offset: 0x0
[INF] Jumping to the first image slot
[INF] Beginning TF-M provisioning
[WRN] TFM_DUMMY_PROVISIONING is not suitable for production! This device is NOT SECURE
[WRN] This device was provisioned with dummy keys. This device is NOT SECURE
```

(continues on next page)

(continued from previous page)

```
[Sec Thread] Secure image initializing!
TF-M isolation level is: 0x00000001
Booting TF-M v1.8.1
```

Note: Some of the messages above are only visible when CMAKE_BUILD_TYPE is set to Debug.

Note: If -DPLATFORM_DEFAULT_PROVISIONING=OFF is set then the provisioning bundle has to be placed on the 0x10022000 address with:

```
$ ./FVP_Corstone_SSE-300_Ethos-U55 -a cpu0*="<path-to-build-directory>/bl2.axf" --data "<path-to-
↪build-directory>/tfm_s_ns_signed.bin"@0x38000000 --data "<path-to-build-directory>/provisioning_
↪bundle.bin"@0x10022000
```

Copyright (c) 2020-2024, Arm Limited. All rights reserved.

Corstone SSE-310 with Ethos-U55/U65 Example Subsystem for Arm Ecosystem FVP and for MPS3 (AN555)

Introduction

Corstone-310 (formerly Corstone-Polaris) is an Arm reference subsystem for secure System on Chips containing an Armv8.1-M Cortex-M85 processor and an Ethos-U55/U65 neural network processor. It is an MPS3 based platform with the usual MPS3 peripherals.

This platform port supports all TF-M regression tests (Secure and Non-secure) with Isolation Level 1 and 2.

Note: For Ethos-U55/U65 IP this platform support only provides base address, interrupt number and an example NPU setup as non-secure, unprivileged.

Note: For Armclang compiler v6.18 or later version is required.

Building TF-M

Follow the instructions in Building instructions.

Build instructions with platform name: arm/mps3/corstone310/an555

-DTFM_PLATFORM=arm/mps3/corstone310/an555

Note: For Ethos-U55/U65 IP this platform support only provides base address, interrupt number and an example NPU setup as non-secure, unprivileged.

Build instructions with platform name: arm/mps3/corstone310/fvp

-DTFM_PLATFORM=arm/mps3/corstone310/fvp

Note: For Ethos-U55/U65 IP this platform support only provides base address, interrupt number and an example NPU setup as non-secure, unprivileged.

Note: The built binaries can be run on the Corstone-310 Ethos-U55/U65 Ecosystem FVP (FVP_SSE310_MPS3). At least Ecosystem FVP version 11.22 is required.

Note: Provisioning bundles can be generated with the -DPLATFORM_DEFAULT_PROVISIONING=OFF flag. The provisioning bundle binary will be generated and it's going to contain the provisioning code and provisioning values.

Note: If -DPLATFORM_DEFAULT_PROVISIONING=OFF and -DTFM_DUMMY_PROVISIONING=ON then the keys in the tf-m/platform/ext/target/arm/mps3/common/provisioning/provisioning_config.cmake and the default MCUBoot signing keys will be used for provisioning.

If -DPLATFORM_DEFAULT_PROVISIONING=OFF and -DTFM_DUMMY_PROVISIONING=OFF are set then unique values can be used for provisioning. The keys and seeds can be changed by passing the new values to the build command, or by setting the -DPROVISIONING_KEYS_CONFIG flag to a .cmake file that contains the keys. An example config cmake file can be seen at tf-m/platform/ext/target/arm/mps3/common/provisioning/provisioning_config.cmake. Otherwise new random values are going to be generated and used. For the image signing the ${MCUBOOT_KEY_S} and ${MCUBOOT_KEY_NS} will be used. These variables should point to .pem files that contain the code signing private keys. The public keys are going to be generated from these private keys and will be used for provisioning. The hash of the public key is going to be written into the provisioning_data.c automatically.

If -DMCUBOOT_GENERATE_SIGNING_KEYPAIR=ON is set then a new mcuboot signing public and private keypair is going to be generated and it's going to be used to sign the S and NS binaries.

The new generated keypair can be found in the <build dir>/bin folder or in the <install directory>/image_signing/keys after installation. The generated provisioning_data.c file can be found at <build directory>/platform/target/provisioning/provisioning_data.c

Note: The provisioning bundle generation depends on pyelftools that's have to be installed:

```
pip3 install pyelftools
```

To run the example code on Corstone SSE-310 with Ethos-U55/U65 Example Subsystem for MPS3 (AN555)

FPGA image is available to download here

If the link above is not working just go to Arm PDH and search for AN555.

To run BL2 bootloader, TF-M example application and tests in the MPS3 board, it is required to have AN555 image in the MPS3 board SD card. The image should be located in <MPS3 device name>/MB/HBI<BoardNumberBoardrevision>/AN555

The MPS3 board tested is HBI0309C.

1. Execute the following command to create the tfm.bin binary which fills the entire available space on the MPS3 onboard QSPI. This way the whole flash content (PS, ITS, OTP, NV coun-

ters) is in a known state. (If anything left in the QSPI, this binary will overwrite it.) Also, MPS3 can only handle SFN 8.3 format, so the binary name must be shortened.:

```
$ cd <build dir>/bin
$ cp tfm_s_ns_signed.bin tfm.bin
$ truncate -s 8M tfm.bin
```

2. Copy bl2.bin and tfm.bin files from build dir to <MPS3 device name>/SOFTWARE/
3. Open <MPS3 device name>/MB/HBI0309C/AN555/images.txt
4. Update the images.txt file as follows:

```
[IMAGES]
TOTALIMAGES: 2

IMAGE0ADDRESS: 0x01_00_1100_0000
IMAGE0UPDATE: RAM
IMAGE0FILE: \SOFTWARE\bl2.bin

IMAGE1ADDRESS: 0x01_00_0000_0000
IMAGE1UPDATE: FORCEQSPI
IMAGE1FILE: \SOFTWARE\tfm.bin
```

5. Close <MPS3 device name>/MB/HBI0309C/AN555/images.txt
6. Unmount/eject the <MPS3 device name> unit
7. Reset the board to execute the TF-M example application
8. After completing the procedure you should be able to visualize on the serial port (baud 115200 8n1) the following messages:

```
[INF] Starting bootloader
[INF] Beginning BL2 provisioning
[WRN] TFM_DUMMY_PROVISIONING is not suitable for production! This device is NOT SECURE
[INF] Image index: 1, Swap type: none
[INF] Image index: 0, Swap type: none
[INF] Bootloader chainload address offset: 0x0
[INF] Jumping to the first image slot
[INF] Beginning TF-M provisioning
[WRN] TFM_DUMMY_PROVISIONING is not suitable for production! This device is NOT SECURE
[WRN] This device was provisioned with dummy keys. This device is NOT SECURE
[Sec Thread] Secure image initializing!
TF-M isolation level is: 0x00000001
Booting TF-M v1.8.1
```

Note: Some of the messages above are only visible when CMAKE_BUILD_TYPE is set to Debug.

Note: If -DPLATFORM_DEFAULT_PROVISIONING=OFF is set then the provisioning bundle has to be placed on the 0x11022400 address by copying provisioning_bundle.bin and renaming it to prv.bin, then extending the images.txt with:

```
IMAGE2UPDATE: RAM
IMAGE2ADDRESS: 0x01_00_1102_2400
IMAGE2FILE: \SOFTWARE\prv.bin
```

To run the example code on Corstone-310 Ethos-U55/U65 Ecosystem FVP

FVP is available to download here

1. Install the FVP
2. Copy bl2.axf and tfm_s_ns_signed.bin files from build dir to <FVP installation path>/models/Linux64_GCC-9.3/
3. Navigate to the same directory and execute the following command to start FVP:

```
$ ./FVP_Corstone_SSE-310 -a cpu0*="bl2.axf" --data "tfm_s_ns_signed.bin"@0x38000000
```

4. After completing the procedure you should be able to see similar messages to this on the serial port (baud 115200 8n1):

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[INF] Starting bootloader
[INF] Beginning BL2 provisioning
[WRN] TFM_DUMMY_PROVISIONING is not suitable for production! This device is NOT SECURE
[INF] Image index: 1, Swap type: none
[INF] Image index: 0, Swap type: none
[INF] Bootloader chainload address offset: 0x0
[INF] Jumping to the first image slot
[INF] Beginning TF-M provisioning
[WRN] TFM_DUMMY_PROVISIONING is not suitable for production! This device is NOT SECURE
[WRN] This device was provisioned with dummy keys. This device is NOT SECURE
[Sec Thread] Secure image initializing!
TF-M isolation level is: 0x00000001
Booting TF-M v1.8.1
```

Note: Some of the messages above are only visible when CMAKE_BUILD_TYPE is set to Debug.

Note: If -DPLATFORM_DEFAULT_PROVISIONING=OFF is set then the provisioning bundle has to be placed on the 0x11022000 address with:

```
$ ./FVP_Corstone_SSE-310 -a cpu0*="<path-to-build-directory>/bl2.axf" --data "<path-to-build-directory>
↪/tfm_s_ns_signed.bin"@0x38000000 --data "<path-to-build-directory>/provisioning_bundle.bin
↪"@0x11022000
```

Copyright (c) 2021-2024, Arm Limited. All rights reserved.

Corstone SSE-315 with Ethos-U65 Example Subsystem for Ecosystem FVP

Introduction

Corstone-315 is an Arm reference subsystem for secure System on Chips containing an Armv8.1-M Cortex-M85 processor and LCM, KMU and SAM IPs. and an Ethos-U65 neural network processor. It is an MPS4 based platform.

This platform port supports all TF-M regression tests (Secure and Non-secure) with Isolation Level 1 and 2.

Note: For Ethos-U65 IP this platform support only provides base address, interrupt number and an example NPU setup as non-secure, unprivileged.

Note: For Armclang compiler v6.18 or later version is required.

Building TF-M

Follow the instructions in Building instructions.

Build instructions with platform name: arm/mps4/corstone315

-DTFM_PLATFORM=arm/mps4/corstone315

Note: The built binaries can be run on the Corstone-315 Ethos-U65 Ecosystem FVP (FVP_Corstone_SSE-315).

Note: If -DTFM_DUMMY_PROVISIONING=ON then the keys in the tf-M/platform/ext/target/arm/mps4/corstone315/provisioning/bundle_cm/cm_provisioning_config.cmake, tf-M/platform/ext/target/arm/mps4/corstone315/provisioning/bundle_dm/dm_provisioning_config.cmake and default MCUBoot signing and encryption keys will be used for provisioning.

If -DTFM_DUMMY_PROVISIONING=OFF is set then unique values can be used for provisioning. The keys and seeds can be changed by passing the new values to the build command, or by setting the -DDM_PROVISIONING_KEYS_CONFIG and -DCM_PROVISIONING_KEYS_CONFIG` flag to the .cmake files which contain the keys. An example config cmake file can be seen at ``tf-m/platform/ext/target/arm/mps4/corstone315/provisioning/bundle_cm/cm_provisioning_config.cmake and tf-m/platform/ext/target/arm/mps4/corstone315/provisioning/bundle_dm/dm_provisioning_config.cmake. Otherwise new random values are going to be generated and used (or default values in some cases). For the image signing the ${MCUBOOT_KEY_S} and ${MCUBOOT_KEY_NS} will be used. These variables should point to .pem files that contain the code signing private keys. The public keys are going to be generated from these private keys and will be used for provisioning. The hash of the public key is going to be written into the provisioning_data.c automatically. For the BL2 image encryption the TFM_BL2_ENCRYPTION_KEY_PATH has to be set to a .bin file which contains the encryption key. (For an example, see tf-m/bl1/bl1_2/bl2_dummy_encryption_key.bin)

If -DMCUBOOT_GENERATE_SIGNING_KEYPAIR=ON is set then a new MCUBoot signing public and private keypair is going to be generated and it's going to be used to sign the S and NS binaries.

The new generated keypair can be found in the <build dir>/bin folder or in the <install directory>/image_signing/keys after installation. The generated provisioning_data.c file can be found at <build directory>/platform/target/provisioning/provisioning_data.c

Note: The provisioning bundle generation depends on pyelftools that's have to be installed:

```
pip3 install pyelftools
```

To run the example code on Corstone-315 Ethos-U65 Ecosystem FVP

FVP is available to download here

1. Install the FVP
2. Copy bl1_1.bin, cm_provisioning_bundle.bin, dm_provisioning_bundle.bin, bl2_signed.bin and tfm_s_ns_signed.bin files from build dir to <FVP installation path>/models/Linux64_GCC-6.4/

3. Navigate to the same directory and execute the following command to start FVP:

```
$ ./FVP_Corstone_SSE-315 --data "bl1_1.bin"@0x11000000
                --data "cm_provisioning_bundle.bin"@0x12024000
                --data "dm_provisioning_bundle.bin"@0x1202aa00
                --data "bl2_signed.bin"@0x12031400
                --data "tfm_s_ns_signed.bin"@0x38000000
```

4. After completing the procedure you should be able to see similar messages to this on the serial port (baud 115200 8n1):

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[INF] Starting TF-M BL1_1
[INF] Beginning provisioning
[INF] TP mode set complete, system will now reset.
[INF] Starting TF-M BL1_1
[INF] Beginning provisioning
[INF] Waiting for CM provisioning bundle
[INF] Enabling secure provisioning mode, system will now reset.
[INF] Starting TF-M BL1_1
[INF] Beginning provisioning
[INF] Waiting for CM provisioning bundle
[INF] Running CM provisioning bundle
[INF] Starting TF-M BL1_1
[INF] Beginning provisioning
[INF] Waiting for DM provisioning bundle
[INF] Enabling secure provisioning mode, system will now reset.
[INF] Starting TF-M BL1_1
[INF] Beginning provisioning
[INF] Waiting for DM provisioning bundle
[INF] Running DM provisioning bundle
[INF] Starting TF-M BL1_1
[INF] Jumping to BL1_2
[INF] starting TF-M bl1_2
[INF] Attempting to boot image 0
[INF] BL2 image decrypted successfully
[INF] BL2 image validated successfully
[INF] Jumping to BL2
[INF] Starting bootloader
[INF] Image index: 1, Swap type: none
[INF] Image index: 0, Swap type: none
[INF] Bootloader chainload address offset: 0x0
[INF] Jumping to the first image slot
Booting TF-M v2.0.0+d09934721
Non-Secure system starting...
```

Note: Some of the messages above are only visible when CMAKE_BUILD_TYPE is set to Debug.

Copyright (c) 2024, Arm Limited. All rights reserved.

Musca-B1 Platform Specifics

DAPLink Firmware

The code on Musca-B1 is running from embedded flash. Make sure that the DAPLink FW for eFlash is downloaded to the board. You can find on the Arm Community page A short description of how to update the DAPLink FW can be found there as well.

Note: Warm reset of eFlash is not supported on Musca_B1. TF-M may not boot after a warm reset. Further information on the hardware limitation can be found on Arm Community page.

Platform pin service

This service is designed to perform secure pin services of the platform (e.g alternate function setting, pin mode setting, etc). The service uses the IOCTL API of TF-M's Platform Service, which allows the non-secure application to make pin service requests on Musca B1 based on a generic service request delivery mechanism.

Copyright (c) 2017-2022, Arm Limited. All rights reserved.

Musca-S1 Platform Specifics

DAPLink Firmware

The code on Musca-S1 is running from on-chip eMRAM (2MB non-volatile bootable memory). Make sure that the DAPLink FW for QSPI/MRAM is downloaded to the board. You can find on the Arm Community page A short description of how to update the DAPLink FW can be found there as well.

Platform pin service

This service is designed to perform secure pin services of the platform (e.g alternate function setting, pin mode setting, etc). The service uses the IOCTL API of TF-M's Platform Service, which allows the non-secure application to make pin service requests on Musca S1 based on a generic service request delivery mechanism.

Copyright (c) 2023, Arm Limited. All rights reserved.

Runtime Security Engine (RSE)

Previously known as Runtime Security Subsystem (RSS).

Runtime Security Engine (RSE)

Introduction

Runtime Security Engine (RSE) is an Arm subsystem that provides a reference implementation of the HES Host in the Arm Confidential Compute Architecture (CCA). It is designed to be integrated into A-profile compute subsystems that implement Arm CCA, where it serves as the Root of Trust.

RSE initially boots from immutable code (BL1_1) in its internal ROM, before jumping to BL1_2, which is provisioned and hash-locked in RSE OTP. The updatable MCUBoot BL2 boot stage is loaded from host system flash into RSE SRAM, where it is authenticated. BL2 loads and authenticates the TF-M runtime into RSE SRAM from host flash. BL2 is also responsible for loading initial boot code into other subsystems within the host.

The RSE platform port supports the TF-M Crypto, TF-M Initial Attestation, Measured Boot and TF-M Platform services along with the corresponding regression tests. It supports the IPC model in multi-core topology with Isolation Level 1 and 2.

Building TF-M

Follow the instructions in *Build instructions*. Build TF-M with platform name: *arm/rse/<rse platform name>*

For example for building RSE for Total Compute platforms: -DTFM_PLATFORM=arm/rse/tc

Signing host images

RSE BL2 can load boot images into other subsystems within the host system. It expects images to be signed, with the signatures attached to the images in the MCUBoot metadata format.

The imgtool Python package can be used to sign images in the required format. To sign a host image using the development key distributed with TF-M, use the following command:

```
imgtool sign \
    -k <TF-M base directory>/bl2/ext/mcuboot/root-RSA-3072.pem \
    --public-key-format full \
    --max-align 8 \
    --align 1 \
    -v "0.0.1" \
    -s 1 \
    -H 0x2000 \
    --pad-header \
    -S 0x80000 \
    --pad \
    -L <load address> \
    <binary infile> \
    <signed binary outfile>
```

The load address is the logical address in the RSE memory map to which BL2 will load the image. RSE FW expects the first host image to be loaded to address 0x70000000 (the beginning of the RSE ATU host access region), and each subsequent host image to be loaded at an offset of 0x1000000 from the previous image. The RSE ATU should be configured to map these logical addresses to the physical addresses in the host system that the images need to be loaded to.

For more information on the imgtool parameters, see the MCUBoot imgtool documentation.

Warning: The TF-M development key must never be used in production. To generate a production key, follow the imgtool documentation.

Running the code

To run the built images, first the ROM image must be created from the bl1_1 binary and the ROM DMA Initial Command Sequence (ICS).:

```
srec_cat \
        bl1_1.bin -Binary      -offset 0x0 \
        rom_dma_ics.bin -Binary -offset 0x1F000 \
        -o rom.bin -Binary
```

Then, the flash image must be created by concatenating the images that are output from the build. To create the flash image, the following fiptool command should be run. fiptool documentation can be found here. Note that an up-to-date fiptool that supports the RSE UUIDs must be used.:

```
fiptool create \
    --align 8192 --rss-bl2     bl2_signed.bin \
    --align 8192 --rss-ns      tfm_ns_signed.bin \
    --align 8192 --rss-s       tfm_s_signed.bin \
    --align 8192 --rss-scp-bl1 <signed Host SCP BL1 image> \
    --align 8192 --rss-ap-bl1  <signed Host AP BL1 image> \
    fip.bin
```

If you already have a fip.bin containing host firmware images, RSE FIP images can be patched in:

```
fiptool update --align 8192 --rss-bl2 bl2_signed.bin fip.bin
fiptool update --align 8192 --rss-ns  tfm_ns.bin fip.bin
fiptool update --align 8192 --rss-s   tfm_s.bin fip.bin
```

If XIP mode is enabled, the following fiptool command should be run to create the flash image:

```
fiptool create \
   --align 8192 --rss-bl2           bl2_signed.bin \
   --align 8192 --rss-ns            tfm_ns_encrypted.bin \
   --align 8192 --rss-s             tfm_s_encrypted.bin \
   --align 8192 --rss-sic-tables-ns tfm_ns_sic_tables_signed.bin \
   --align 8192 --rss-sic-tables-s  tfm_s_sic_tables_signed.bin \
   --align 8192 --rss-scp-bl1       <signed Host SCP BL1 image> \
   --align 8192 --rss-ap-bl1        <signed Host AP BL1 image> \
   fip.bin
```

Once the FIP is prepared, a host flash image can be created using srec_cat:

```
srec_cat \
       fip.bin -Binary -offset 0x0 \
       -o host_flash.bin -Binary
```

If GPT support is enabled, and a host fip.bin and fip_gpt.bin has been obtained, RSE images can be inserted by first patching the host FIP and then inserting that patched FIP into the GPT image:

```
sector_size=$(gdisk -l fip_gpt.bin | grep -i "sector size (logical):" | \
           sed 's/.*logical): \([0-9]*\) bytes/\1/')

fip_label=" FIP_A$"
fip_start_sector=$(gdisk -l fip_gpt.bin | grep "$fip_label" | awk '{print $2}')
fip_sector_am=$(gdisk -l fip_gpt.bin | grep "$fip_label" | awk '{print $3 - $2}')

dd if=fip.bin of=fip_gpt.bin bs=$sector_size seek=$fip_start_sector \
   count=$fip_sector_am conv=notrunc

fip_label = " FIP_B$"
fip_start_sector = $(gdisk -l fip_gpt.bin | grep "$fip_label" | awk '{print $2}')
fip_sector_am = $(gdisk -l fip_gpt.bin | grep "$fip_label" | awk '{print $3 - $2}')

dd if=fip.bin of=fip_gpt.bin bs=$sector_size seek=$fip_start_sector \
   count=$fip_sector_am conv=notrunc
```

To patch a fip_gpt.bin without having an initial fip.bin, the FIP can be extracted from the GPT image using the following commands (and can then be patched and reinserted using the above commands):

```
sector_size=$(gdisk -l fip_gpt.bin | grep -i "sector size (logical):" | \
           sed 's/.*logical): \([0-9]*\) bytes/\1/')

fip_label=" FIP_A$"
fip_start_sector=$(gdisk -l fip_gpt.bin | grep "$fip_label" | awk '{print $2}')
fip_sector_am=$(gdisk -l fip_gpt.bin | grep "$fip_label" | awk '{print $3 - $2}')

dd if=fip_gpt.bin of=fip.bin bs=$sector_size skip=$fip_start_sector \
   count=$fip_sector_am conv=notrunc
```

Once the fip_gpt.bin is prepared, it is placed at the base of the host flash image:

```
srec_cat \
       fip_gpt.bin -Binary -offset 0x0 \
       -o host_flash.bin -Binary
```

The RSE ROM binary should be placed in RSE ROM at 0x11000000 and the host flash binary should be placed at the base of the host flash. For the TC platform, this is at 0x80000000.

The RSE OTP must be provisioned. On a development platform with TFM_DUMMY_PROVISIONING enabled, BL1_1 expects provisioning bundles to be preloaded into SRAM. Preload encrypted_cm_provisioning_bundle_0.bin to the base of VM0, and encrypted_dm_provisioning_bundle.bin to the base of VM1.

If TFM_DUMMY_PROVISIONING is disabled and provisioning is required, then BL1_1 will first wait for the TP mode to be set by a debugger (setting the tp_mode variable in the current stack frame is easiest). BL1_1 will then wait for provisioning bundles to be loaded to VM0 and VM1 in the same way as when TFM_DUMMY_PROVISIONING is enabled, except that it will not automatically perform the reset once each provisioning state is complete. For more details about provisioning flows, see *RSE provisioning*.

Copyright (c) 2022-2023, Arm Limited. All rights reserved.

RSE communication design

The RSE communication protocol is designed to be a lightweight serialization of the psa_call() API through a combination of in-band MHU (Message Handling Unit) transport and parameter-passing through shared memory.

Message format

To call an RSE service, the client must send a message in-band over the MHU sender link to RSE and wait for a reply message on the MHU receiver (either by polling the MHU or waiting for interrupt). The messages are defined as packed C structs, which are serialized in byte-order over the MHU links.

Messages encoding a psa_call() to RSE take the form:

```
__PACKED_STRUCT serialized_psa_msg_t {
    struct serialized_rse_comms_header_t header;
    __PACKED_UNION {
        struct rse_embed_msg_t embed;
        struct rse_pointer_access_msg_t pointer_access;
    } msg;
};
```

Replies from RSE take the form:

```
__PACKED_STRUCT serialized_psa_reply_t {
    struct serialized_rse_comms_header_t header;
    __PACKED_UNION {
        struct rse_embed_reply_t embed;
        struct rse_pointer_access_reply_t pointer_access;
    } reply;
};
```

All messages (calls and replies), in all protocols carry the following header:

```
__PACKED_STRUCT serialized_rse_comms_header_t {
    uint8_t protocol_ver;
    uint8_t seq_num;
    uint16_t client_id;
};
```

The client_id can be used by the caller to identify different clients at the endpoint for access control purposes. It is combined with an RSE-internal identifier for the endpoint to create the PSA Client ID for the call.

The sequence number, seq_num, is returned in the reply message to allow the client to identify which message is being responded to, since replies may be out-of-order. The sender is free to assign sequence numbers using a scheme of its choice, and it is the sender's responsibility to ensure that two messages with the same sequence number are not in simultaneous flight.

The protocol_ver identifies which message protocol is being used. There are two protocols currently supported, "embed" (protocol_ver=0) and "pointer access" (protocol_ver=1).

Embed protocol

The embed protocol embeds the psa_call iovecs into the message sent over the MHU. It has the following format:

```
__PACKED_STRUCT rse_embed_msg_t {
    psa_handle_t handle;
    uint32_t ctrl_param;
    uint16_t io_size[PSA_MAX_IOVEC];
    uint8_t payload[RSE_COMMS_PAYLOAD_MAX_SIZE];
};
```

The handle is the psa_call handle parameter and the ctrl_param packs the type, in_len and out_len parameters of psa_call into one parameter.

The io_size array contains the sizes of the up to PSA_MAX_IOVEC (4) iovecs, in order, with the invec sizes before the outvec sizes.

The payload array then contains the invec data packed contiguously in order. The length of this parameter is variable, equal to the sum of the invec lengths in io_size. The caller does not need to pad the payload to the maximum size. The maximum payload size for this protocol, RSE_COMMS_PAYLOAD_MAX_SIZE, is a build-time option.

Replies in the embed protocol take the form:

```
__PACKED_STRUCT rse_embed_reply_t {
    int32_t return_val;
    uint16_t out_size[PSA_MAX_IOVEC];
    uint8_t payload[RSE_COMMS_PAYLOAD_MAX_SIZE];
};
```

The return_val is the return value of the psa_call() invocation, the out_size is the (potentially updated) sizes of the outvecs and the payload buffer contains the outvec data serialized contiguously in outvec order.

Pointer access protocol

The pointer access protocol passes the psa_call iovecs as pointers to shared memory using the following MHU message format:

```
__PACKED_STRUCT rse_pointer_access_msg_t {
    psa_handle_t handle;
    uint32_t ctrl_param;
    uint32_t io_sizes[PSA_MAX_IOVEC];
    uint64_t host_ptrs[PSA_MAX_IOVEC];
};
```

The handle, ctrl_param and io_sizes have the same definition as in the embed protocol.

The host_ptrs point to each of the up to PSA_MAX_IOVEC iovec buffers in host system memory. It is the caller's responsibility to write the invec data to the invec pointers before sending the message, and ensure that the memory pointed-to remains valid for the duration of the call.

The reply message has the form:

```
__PACKED_STRUCT rse_pointer_access_reply_t {
    int32_t return_val;
    uint32_t out_size[PSA_MAX_IOVEC];
};
```

The return_val and out_size have the same meaning as in the embed protocol.

RSE writes the outvec data to the pointers supplied in the call message prior to sending the MHU reply message, so no further payload is sent in the reply message.

Implementation structure

The RSE side of the communication implementation is located in platform/ext/target/arm/rse/common/rse_comms. The implementation is structured as follows:

- rse_comms.c: Implements the TF-M RPC layer using RSE comms implementation.
- rse_comms_hal.c: Abstracts MHU message sending and receiving.
- rse_comms_protocol.c: The common part of the RSE comms protocol.
- rse_comms_protocol_embed.c: The embed RSE comms protocol.
- rse_comms_protocol_protocol_access.c: The pointer access RSE comms protocol.
- rse_comms_atu.c: Allocates and frees ATU regions for host pointer access.
- rse_comms_permissions_hal.c: Checks service access permissions and pointer validity.

A reference implementation of the client side of the RSE comms is available in the Trusted Firmware-A repository.

Copyright (c) 2022-2023, Arm Limited. All rights reserved.

RSE key management

The RSE has a set of hardware components that act together to prevent hardware protected keys being disclosed to compromised software. This chain involves the LifeCycle Manager “LCM”, the Key Management Unit “KMU”, and the cryptographic accelerator in the integration layer of the RSE (which in reference RSE builds is a CryptoCell-3XX series accelerator).

Hardware protected keys (henceforth “Hardware Keys”) are stored in the LCM, in protected keyslots. On production-mode “PCI” chips these slots are write-only. They are protected during provisioning from disclosure by Secure Provisioning “SP” mode disabling debug, and the encryption with either the RTL key or the CM provisioning key. On boot, the LCM checks the integrity of the stored hardware keys by comparing the amount of zeroes with a stored zero-count which, because of the property of the OTP that does not allow 1-bits to be set to 0-bits, allows detection of any changes in the keys. If the check succeeds it exports the keys to the KMU over a private memory bus (one that is not accessible programs running on the RSE’ CPU or other controllers on the main bus such as the DMA).

The KMU has between 2 and 32 hardware keyslots, which are 256 bit (32 byte) in size. Between 0 and 8 of these are hardware keyslots, and correspond to the hardware keyslots in the LCM. These slots are filled by the key export from the LCM, and have their permissions preconfigured so they are not readable or writable by software. Software KMU keyslots (keyslots that are not defined in hardware as hardware keyslots) can be read and written, and then subsequently locked to prevent reading and writing, so that they behave like hardware keyslots. This allows trusted immutable firmware to set up keys for less-trusted later stage code, which can be used but cannot be read. KMU keyslots can be exported over another private bus to the cryptographic accelerator. Hardware keyslots can be invalidated, making them unusable until a hard reset.

The cryptographic accelerator receives the key into a write-only key register, and then can use it to perform cryptographic operations. For the CryptoCell-3XX series, this is limited to AES. As

with the previous stages, software cannot retrieve the key, only use it. Note that currently this can only be used with the cc3xx_rom_lib driver, not the cryptocell-312-runtime driver, as the latter requires modification. Note that this path is the only way to use hardware protected keys on the RSE, they cannot be used by software cryptographic libraries that require key material to be accessible by software.

The RSE uses this functionality to allow key derivation (based on NIST SP800-108 with an RSA-based PRF that can utilise the hardware keys as described above) from the HUK and GUK. This allows the derivation of the CCA platform attestation key / delegated attestation root key without allowing access to the GUK directly. As a further security measure, these keys are derived by ROM/OTP code, and the HUK and GUK are then invalidated so they cannot be used by the runtime firmware and BL2.

Copyright (c) 2022-2023, Arm Limited. All rights reserved.

RSE provisioning

Provisioning theory

The LifeCycle Manager (LCM) controls access to the RSE OTP, and includes a state-machine that controls Lifecycle-state (LCS) transitions. The LCM is derived from the OTP management and state machine subsystems of the CryptoCell-3XX series accelerators, and will be familiar to those who have worked with them.

When the chip hasn't been provisioned, the OTP is blank, which means the LCM is in "Virgin" Test/Production mode "TP mode" state. The first step for provisioning must be to set the LCM to either test-chip mode "TCI" or production-chip mode "PCI". In TCI mode the RTL key is masked to avoid disclosure, several OTP fields are changed from write-only to read-write, to aid in debugging, and debugging is not limited in secure provisioning mode (though the RTL key reads as zero). If TFM_DUMMY_PROVISIONING is enabled in cmake (which it is by default) then the chip will be set to TCI mode. If this option is not enabled, execution will pause to allow the setting to be set by a debugger.

Once the TP mode has been set, the chip is then is Chip Manufacturer provisioning state "CM". This mode is intended for the provisioning of the HUK, GUK, CM provisioning key, CM code-encryption key, the root-of-trust public key and the CM config. To provision these fields, The firmware must first receive a provisioning bundle via a debugger. This bundle must be placed at the start of VM0 + an offset the size of the OTP DMA ICS (usually 0x400). This bundle contains the keys and also code to perform the provisioning such as a driver for the LCM, and a function to randomly generate the HUK via the CryptoCell TRNG. The chip must then enter secure provisioning mode by setting the SP_ENABLE register. This causes a reset (but does not clear the RSE SRAMs), and allows access to the RTL key by exporting it to the KMU, though in secure provisioning mode the ability to debug the RSE is disabled, to prevent disclosure of the decrypted provisioning bundle values. The RSE will then decrypt and authenticate the bundle using the RTL key. Under TCI mode the RTL key is zeroed, the bundle generation tool must use a zeroed key to encrypt and sign the bundle. Once the CM provisioning bundle has been unpacked, the RSE will execute the code which will provision the CM provisioning data into OTP. The RSE must be cold-reset, which will disable secure provisioning mode. If TFM_DUMMY_PROVISIONING is enabled the reset will happen automatically, else the external provisioning device should read the provisioning state from the GPIO/PSI (which is set via the rse_sysctrl register) and perform the reset.

After the cold reset, the RSE will automatically transition to Device Manufacturer provisioning state "DM" as the LCM hardware state-machine reads the values of the cm_config_1 and cm_config_2 fields as non-zero. This state is designed to provision the DM provisioning key, the DM code-encryption key and the DM config. The procedure follows the same steps as the CM provisioning flow, with the exception that the bundle will now be encrypted and signed using the CM provisioning key and must be placed at the base of VM1. As before, once the provisioning

bundle has been unpacked/run, the RSE must either be cold-reset or will perform this automatically.

After the cold reset, the device will now be in Secure Enable "SE" mode, due to the dm_config_1 field being non-zero. Debug may be limited based on the hardware DCU mask for SE state. Provisioning will not be run on boot.

Practical RSE provisioning

The RSE buildsystem produces two provisioning bundles (containing both code and data), and then encrypts and signs them with the RTL key to produce encrypted_cm_provisioning_bundle.bin and encrypted_dm_provisioning_bundle.bin.

Note: Currently encrypted provisioning bundles are not supported due to a lack of AEAD encryption support in the cc312-rom-lib driver. The encrypted_*_provisioning_bundle.bin files should still be used, but note that their contents are not encrypted.

On first boot, the RSE is in Virgin state. If the RSE firmware was built with TFM_DUMMY_PROVISIONING enabled then it will automatically set the chip to TCI mode and cold-reset. Production ROM implementations must disable TFM_DUMMY_PROVISIONING, which will cause RSE to loop in the ROM until either TCI or PCI mode is set with a debugger. It is possible to set the TP mode in the LCS registers directly, however it may be easier to set the tp_mode variable in the frame where RSE is looping, at which point the loop will exit and the TP mode will be set by the ROM code.

On non-virgin boot in CM lifecycle state, RSE checks the start of VM0 for the magic constant 0xC0DEFEED, which is required to be the first word in the CM provisioning bundle. There is also a second check for a constant at the end of the bundle to ensure the bundle has finished writing. The RSE will perform this check in a loop until a bundle is found.

This procedure is repeated for DM LCS, except that the magic constant is 0xBEEFFEED and the bundle must be loaded to the base of VM1. Note that the size of RSE memory may vary depending on implementation, so the load address of the DM bundle may change.

In production systems it is intended that these bundles are loaded by a debugger, but for development systems it may be too onerous to perform this procedure, particularly if the system is one that has ephemeral OTP such as an FVP. The preferred solution is to preload to the provisioning bundles into VM0 and VM1 as part of the image loading, which is supported on FVPs but may not be on other systems. An alternative solution is to perform provisioning manually once, and then to save the state of the OTP in SE LCS and then preload that on subsequent boots.

RSE provisioning GPIO signalling

The state of the RSE ROM boot/provisioning flow is signalled outside of the RSE subsystem via the GPIOs as part of the Persistent State Interface (PSI). The PSI signals the lifecycle state as a hardware signal, but additionally the software can signal over the PSI by setting the rse_sysctrl register.

The boot state is encoded in the lowest 4 bits of the rse_sysctrl register, and has meaning as follows:

Signal	State
0x0	RSE cold boot default
0x1	Virgin chip idle, ready to set PCI/TCI mode
0x2	CM LCS idle, waiting for CM provisioning bundle
0x3	RMA LCS idle
0x4	CM secure provisioning started, secure provisioning mode enabled
0x5	CM secure provisioning failed due to bundle authentication error
0x6	CM secure provisioning failed due to other error
0x7	CM secure provisioning succeeded
0x8	DM LCS idle, waiting for CM provisioning bundle
0x9	DM secure provisioning started, secure provisioning mode enabled
0xa	DM secure provisioning failed due to bundle authentication error
0xb	DM secure provisioning failed due to other error
0xc	DM secure provisioning succeeded
0xd	SE LCS standard boot

Copyright (c) 2022-2024, Arm Limited. All rights reserved.

RSE to RSE routing

In multi-RSE implementations, RSE uses a routing protocol to allow messages to be sent from any RSE to any other RSE, regardless of if there is a physical link between them. This may involve an intermediate RSE forwarding the message.

A routing table is used to indicate which is the next-hop for any message. This table is indexed by the destination RSE and contains an index indicating which hardware link should be used to send the message.

The table is located in OTP, which allows for it to be provisioned once the chip topology is known. This also allows BL1_2 code to be identical across all RSEs in a system.

The routing table is automatically constructed by the build system. Each platform must specify a .tgf (trivial graph format) file which describes the system topology, which is then used by the provisioning scripts to calculate the next-hop for each RSE to every other RSE. This generates individual provisioning bundles for each RSE containing their routing tables.

RSE System Topology Map

The system topology map is described as a .tgf file. The MULTI_RSE_TOPOLOGY_FILE cmake variable configures which system topology map tgf file will be used by the build system. This variable should be set in RSE subplatform default config for any subplatform which contains multiple RSEs.

Documentation of the TGF format can be found at https://en.wikipedia.org/wiki/Trivial_Graph_Format#Format.

In the system topology map, each node represents an RSE. The Label portion of the node is optional, but it is recommended to use it to describe which RSE the node represents. Each edge represents a physical link between the corresponding RSEs. The edges are directional, with the first index representing the RSE whose link IDs are described by the edge (henceforth referred to as the Local RSE) and the second index representing the RSE which is being communicated with (henceforth referred to as the Remote RSE).

An example graph node line is:

```
0 Primary RSE
```

Because knowing that a link exists does not encode which link should be used, the link ID is encoded as the name of the edge. The format of this name must be "Send <n> Receive <m>" where <n> is the index of the link which is used by the Local RSE to send to the Remote RSE, and <m> is the index of the link which is used by the Local RSE to receive from the Remote RSE.

An example graph edge line is:

```
0 1 Send 1 Receive 2
```

In this example, the Local RSE has index 0 and the Remote RSE has index 1. If the Local RSE wants to send a message to the Remote RSE, it should send the message via the link with index 1. If the Local RSE wants to listen for messages from the Remote RSE, it should listen on the link with index 2.

The following is an example topology map for a system with 4 RSEs, which are all connected to each other:

```
0 RSE 0
1 RSE 1
2 RSE 2
3 RSE 3

0 1 Send 1 Receive 1
0 2 Send 2 Receive 2
0 3 Send 3 Receive 3

1 0 Send 0 Receive 0
1 2 Send 2 Receive 2
1 3 Send 3 Receive 3

2 0 Send 0 Receive 0
2 1 Send 1 Receive 1
2 3 Send 3 Receive 3

3 0 Send 0 Receive 0
3 1 Send 1 Receive 1
3 2 Send 2 Receive 2
```

The following is a "hub-and-spoke" system topology with 5 RSEs:

```
0 Hub
1 Spoke 0
2 Spoke 1
3 Spoke 2
4 Spoke 3

0 1 Send 0 Receive 0
0 2 Send 1 Receive 1
0 3 Send 2 Receive 2
0 4 Send 3 Receive 3

1 0 Send 0 Receive 0
2 0 Send 0 Receive 0
3 0 Send 0 Receive 0
4 0 Send 0 Receive 0
```

RSE Routing Table

The RSE routing table is provisioned into OTP. It is provisioned as two distinct tables, as "send" table and a "receive" table. Each table entry is a uint32_t, which is used as an index into the link table in order to decide which link to send/receive on. The amount of elements in the routing table is the same as the number of RSEs in the system.

In order to find the link that should be used, the destination RSE should be used as an index in the correct routing table. The value in the routing table should then be used to select the link.

The size of the routing tables is 4 bytes * 2 * the number of RSEs in the system.

The value in the table which corresponds to the current RSE must be 0. It is invalid to send a message to the same RSE that is sending the message. 0 is also a valid link index.

Each RSE in the system is provisioned with different routing tables. The routing tables for each RSE are generated by running a shortest-path algorithm on the graph between the RSE whose table is being generated and every other RSE.

The link table for RSE 2 in the first example above (fully connected) is:

```
Send    = {0, 1, 0, 3}
Receive = {0, 1, 0, 3}
```

The link table for RSE 1 in the Second example above (hub-and-spoke) is:

```
Send    = {0, 0, 0, 0}
Receive = {0, 0, 0, 0}
```

Copyright (c) 2024, Arm Limited. All rights reserved.

RSE ROM Releases

RSE ROM Release 2024-04-08

Contents

- *Features*
- *Known errata*
- *TF-M version*
- *Tested Environment*
- *Build command*

Features

- A ROM bootloader for RSE that supports chainloading a second immutable boot stage (BL1-2) from OTP.
- Integrity protection of the BL1-2 boot image using a provisioned SHA-256 hash of the image.
- Hardware-accelerated crypto operations using the built-in CryptoCell-3xx ROM driver.
- Support for hardware lifecycle management and provisioning using code and values loaded through a debugger.
- A DMA Initial Command Sequence placed at offset 0x1F000 in the ROM.
- Support for secure provisioning, using AEAD encryption with keys derived from the RTL key.
- Allows for setup of SAM configuration with provisioned values before CPU is started via the DMA ICS.
- Uses TRAM for BL1_1 data sections.

Known errata

- The build instructions for this release initially contained an incorrect address for the DMA ICS concatenation with the ROM code. The build instructions in this document have been updated and are now correct. The correct address is 0x1F000, instead of 0x1E000. Some distributed ROM binaries have also been impacted, and hence will not boot.

  If a ROM binary does not boot, it can be checked for this error using the command:

  ```
  xxd -s 0x1F000 rom.bin
  ```

  If there is no output, or the output is all zero-words, then the ROM binary is affected by the issue. It is possible to fix a ROM binary affected by the issue by running the following command:

  ```
  dd if=rom.bin bs=1k skip=120 seek=124 count=4 of=rom.bin
  ```

TF-M version

The RSE ROM image should be generated from TF-M commit hash:

```
2ca8c58dc2feddc93b87585ffc07e4d169f54278
```

Tested Environment

- arm-none-eabi-gcc –version | head -n 1

```
arm-none-eabi-gcc (GNU Arm Embedded Toolchain 10.3-2021.10) 10.3.1 20210824 (release)
```

- python3 –version

```
Python 3.11.6
```

- python3 -m pip list

```
Package                    Version
-------------------------- ---------
attrs                       23.2.0
cbor2                        5.6.2
cffi                        1.16.0
click                       8.1.7
cryptography                 42.0.5
ecdsa                       0.18.0
imgtool                      2.0.0
intelhex                     2.3.0
Jinja2                      3.1.3
jsonschema                   4.21.1
jsonschema-specifications 2023.12.1
kconfiglib                  14.1.0
MarkupSafe                   2.1.5
networkx                     3.3
pip                         23.2
pyasn1                       0.6.0
pycparser                    2.22
pyhsslms                     2.0.0
PyYAML                       6.0.1
referencing                 0.34.0
rpds-py                     0.18.0
setuptools                  68.1.2
six                         1.16.0
```

Build command

The RSE ROM image should be generated with the following build commands:

```
python3 -m venv ./venv
source ./venv/bin/activate

python3 -m pip install -r ./tools/requirements.txt

cmake -S . -B build -DTFM_PLATFORM=arm/rse/tc \
    -DTFM_TOOLCHAIN_FILE=toolchain_GNUARM.cmake \
    -DCMAKE_BUILD_TYPE=minsizerel \
    -DRSE_ENABLE_TRAM=ON \
    -DRSE_OTP_TRNG=ON \
    -DTFM_DUMMY_PROVISIONING=OFF \
    -DRSE_XIP=ON \
    -DTFM_PARTITION_DPE=OFF

cmake --build build -- install

srec_cat build/bin/bl1_1.bin  -Binary -offset 0x0 \
    build/bin/rom_dma_ics.bin -Binary -offset 0x1F000 \
    -o rse_rom_2024-04-08.bin -Binary

truncate --size 131072 rse_rom_2024-04-08.bin
```

Copyright (c) 2024, Arm Limited. All rights reserved.

Copyright (c) 2024, Arm Limited. All rights reserved.

RSE also includes the following extra partitions:

- Authenticated Debug Access Control (ADAC)
- Measured boot partition
- Delegated attestation partition
- DICE Protection Environment partition

Copyright (c) 2022-2024, Arm Limited. All rights reserved.

Copyright (c) 2022-2024, Arm Limited. All rights reserved.

ArmChina platforms

Arm China Alcor with Cortex-M52 Example Subsystem for MPS3 (AN557)

Introduction

Alcor is an ArmChina reference subsystem for secure System on Chips containing an Armv8.1-M Cortex-M52 processor. It is an MPS3 based platform with the usual MPS3 peripherals.

This platform port supports all TF-M regression tests (Secure and Non-secure) with Isolation Level 1 and 2.

Note: For Armclang compiler v6.18 or later version is required.

Building TF-M

Follow the instructions in Building instructions.

Build instructions with platform name: armchina/mps3/alcor/an557

-DTFM_PLATFORM=armchina/mps3/alcor/an557

Note: Provisioning bundles can be generated with the -DPLATFORM_DEFAULT_PROVISIONING=OFF flag. The provisioning bundle binary will be generated and it's going to contain the provisioning code and provisioning values.

Note: If -DPLATFORM_DEFAULT_PROVISIONING=OFF and -DTFM_DUMMY_PROVISIONING=ON then the keys in the tf-m/platform/ext/target/armchina/mps3/common/provisioning/provisioning_config.cmake and the default MCUBoot signing keys will be used for provisioning.

If -DPLATFORM_DEFAULT_PROVISIONING=OFF and -DTFM_DUMMY_PROVISIONING=OFF are set then unique values can be used for provisioning. The keys and seeds can be changed by passing the new values to the build command, or by setting the -DPROVISIONING_KEYS_CONFIG flag to a .cmake file that contains the keys. An example config cmake file can be seen at tf-m/platform/ext/target/armchina/mps3/common/provisioning/provisioning_config.cmake. Otherwise new random values are going to be generated and used. For the image signing the ${MCUBOOT_KEY_S} and ${MCUBOOT_KEY_NS} will be used. These variables should point to .pem files that contain the code signing private keys. The public keys are going to be generated from these private keys and will be used for provisioning. The hash of the public key is going to be written into the provisioning_data.c automatically.

If -DMCUBOOT_GENERATE_SIGNING_KEYPAIR=ON is set then a new mcuboot signing public and private keypair is going to be generated and it's going to be used to sign the S and NS binaries.

The new generated keypair can be found in the <build dir>/bin folder or in the <install directory>/image_signing/keys after installation. The generated provisioning_data.c file can be found at <build directory>/platform/target/provisioning/provisioning_data.c

---

Note: The provisioning bundle generation depends on pyelftools that's have to be installed:

```
pip3 install pyelftools
```

To run the example code on Arm China Alcor with Cortex-M52 Example Subsystem for MPS3 (AN557)

To run BL2 bootloader, TF-M example application and tests in the MPS3 board, it is required to have AN557 image in the MPS3 board SD card. The image should be located in <MPS3 device name>/MB/HBI<BoardNumberBoardrevision>/AN557

The MPS3 board tested is HBI0309C.

1. Copy bl2.bin and tfm_s_ns_signed.bin files from build dir to <MPS3 device name>/SOFTWARE/
2. Rename tfm_s_ns_signed.bin to tfm.bin (Filename should not be longer than 8 charachters.)
3. Open <MPS3 device name>/MB/HBI0309C/AN557/images.txt
4. Update the images.txt file as follows:

```
TITLE: Arm MPS3 FPGA prototyping board Images Configuration File

[IMAGES]
TOTALIMAGES: 2                  ;Number of Images (Max: 32)

IMAGE0UPDATE: AUTO              ;Image Update:NONE/AUTO/FORCE
IMAGE0ADDRESS: 0x00000000       ;Please select the required executable program
IMAGE0FILE: \SOFTWARE\bl2.bin
IMAGE1UPDATE: FORCEQSPI
IMAGE1ADDRESS: 0x00000000
IMAGE1FILE: \SOFTWARE\tfm.bin
```

5. Close <MPS3 device name>/MB/HBI0309C/AN557/images.txt
6. Unmount/eject the <MPS3 device name> unit
7. Reset the board to execute the TF-M example application
8. After completing the procedure you should be able to visualize on the serial port (baud 115200 8n1) the following messages:

```
[INF] Starting bootloader
[INF] Beginning BL2 provisioning
[WRN] TFM_DUMMY_PROVISIONING is not suitable for production! This device is NOT SECURE
[INF] Image index: 1, Swap type: none
[INF] Image index: 0, Swap type: none
[INF] Bootloader chainload address offset: 0x0
[INF] Jumping to the first image slot
[INF] Beginning TF-M provisioning
[WRN] TFM_DUMMY_PROVISIONING is not suitable for production! This device is NOT SECURE
[WRN] This device was provisioned with dummy keys. This device is NOT SECURE
[Sec Thread] Secure image initializing!
TF-M isolation level is: 0x00000001
Booting TF-M v2.0.0
```

Note: Some of the messages above are only visible when CMAKE_BUILD_TYPE is set to Debug.

Note: If -DPLATFORM_DEFAULT_PROVISIONING=OFF is set then the provisioning bundle has to be placed on the 0x10022400 address by copying encrypted_provisioning_bundle.bin and renaming it to prv.bin, then extending the images.txt with:

```
IMAGE2UPDATE: AUTO
IMAGE2ADDRESS: 0x00022400
IMAGE2FILE: \SOFTWARE\prv.bin
```

Copyright (c) 2020-2023, Arm Limited. All rights reserved. Copyright (c) 2024, ArmChina. All rights reserved.

Copyright (c) 2024, ArmChina. All rights reserved.

Cypress platform

Cypress PSoC64

Cypress PSoC64 Specifics

Prerequisites

PSoC64 must first be provisioned with SecureBoot firmware and a provisioning packet containing policy and secure keys. Please refer to the guide at https://www.cypress.com/documentation/software-and-drivers/psoc-64-secure-mcu-secure-boot-sdk-user-guide Use the following policy file for provisioning and signing: policy_multi_CM0_CM4_tfm.json

Please make sure you have all required software installed as explained in the TF-M getting started.

Please also make sure that all the source code are fetched by following *general building instruction*.

Install CySecureTools or update it to the newest version (2.0.0 at the time of writing)

```
pip3 install --upgrade --force-reinstall cysecuretools
```

PyOCD is installed by CySecureTools as a dependency.

For more details please refer to CySecureTools page.

Install OpenOCD with PSoC6 support. Download the latest revision 4.1.0 or newer from: https://github.com/cypresssemiconductorco/openocd/releases

Lastly, the PSoC64 board KitProg firmware needs to be 2.00.744 or greater. Please use Cypress Modus Toolbox or CyProgrammer to update it if needed.

Building Multi-Core TF-M on Cypress PSoC64

Configuring the build

The build configuration for TF-M is provided to the build system using command line arguments. Required arguments are noted below.

- – -DTFM_PLATFORM=cypress/psoc64
  – Specifies target platform name psoc64
- – -DTFM_TOOLCHAIN_FILE=<path to toolchain file>
  – Specifies the compiler toolchain The possible values are:
    * <TFM root dir>/toolchain_ARMCLANG.cmake
    * <TFM root dir>/toolchain_GNUARM.cmake
    * <TFM root dir>/toolchain_IARARM.cmake

see *Build configuration* for more information.

Build Instructions

The following instructions build multi-core TF-M without regression test suites in Isolation Level 1 on Linux. Both the compiler and the debugging type can be changed to other configurations listed above.

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

mkdir <build folder>
pushd <build folder>
cmake -DTFM_PLATFORM=cypress/psoc64 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      ../
popd
cmake --build <build folder> -- -j VERBOSE=1
```

The following instructions build multi-core TF-M with regression test suites in Isolation Level 1 on Linux. Both the compiler and the debugging type can be changed to other configurations listed above.

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

mkdir <build folder>
pushd <build folder>
cmake -DTFM_PLATFORM=cypress/psoc64 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTEST_S=ON -DTEST_NS=ON \
      ../
popd
cmake --build <build folder> -- -j VERBOSE=1
```

The following instructions build multi-core TF-M with PSA API test suite for the attestation service in Isolation Level 1 on Linux. Both the compiler and the debugging type can be changed to other configurations listed above.

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

mkdir <build folder>
pushd <build folder>
cmake -DTFM_PLATFORM=cypress/psoc64 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTEST_PSA_API=INITIAL_ATTESTATION \
      ../
popd
cmake --build <build folder> -- -j VERBOSE=1
```

The following instructions build multi-core TF-M without regression test suites in Isolation Level 2 on Linux. Both the compiler and the debugging type can be changed to other configurations listed above.

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

mkdir <build folder>
pushd <build folder>
cmake -DTFM_PLATFORM=cypress/psoc64 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTFM_ISOLATION_LEVEL=2 \
      ../
popd
cmake --build <build folder> -- -j VERBOSE=1
```

The following instructions build multi-core TF-M with regression test suites in Isolation Level 2 on Linux. Both the compiler and the debugging type can be changed to other configurations listed above.

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

mkdir <build folder>
pushd <build folder>
cmake -DTFM_PLATFORM=cypress/psoc64 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTFM_ISOLATION_LEVEL=2 \
      -DTEST_S=ON -DTEST_NS=ON \
      ../
```

(continues on next page)

(continued from previous page)

```
popd
cmake --build <build folder> -- -j VERBOSE=1
```

The following instructions build multi-core TF-M with PSA API test suite for the protected storage service in Isolation Level 2 on Linux. Both the compiler and the debugging type can be changed to other configurations listed above.

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

mkdir <build folder>
pushd <build folder>
cmake -DTFM_PLATFORM=cypress/psoc64 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTFM_ISOLATION_LEVEL=2 \
      -DTEST_PSA_API=PROTECTED_STORAGE \
      ../
popd
cmake --build <build folder> -- -j VERBOSE=1
```

Signing

Signing keys

Copy secure keys used in the board provisioning process to platform/ext/target/cypress/psoc64/security/keys:

The keys included in the repository are for reference and development only. DO NOT USE THESE KEYS IN ANY ACTUAL DEPLOYMENT!

Note: provisioned board in SECURE state is required, otherwise refer to Cypress documentation for details on the provisioning process. https://www.cypress.com/documentation/software-and-drivers/psoc-64-secure-mcu-secure-boot-sdk-user-guide

If the board was previously provisioned with signing keys and policy, copy secure signing keys used in the board provisioning process to platform/ext/target/cypress/psoc64/security/keys:

TFM_S_KEY.json
: Private OEM key for signing CM0P image

TFM_S_KEY_PRIV.pem
: Private OEM key for signing CM0P image in PEM format

TFM_NS_KEY.json
: Private OEM key for signing CM4 image

TFM_NS_KEY_PRIV.pem
: Private OEM key for signing CM4 image in PEM format

Alternatively, you can generate a new set of signing keys using cysecuretools create-keys command and provision the keys to the board, if the previously provisioned policy allows board's re-provisioning.

Initialize cysecuretools environment:

```
cd platform/ext/target/cypress/psoc64/security
cysecuretools -t cy8ckit-064s0s2-4343w init
```

Generate a new set of keys:

```
cysecuretools -t cy8ckit-064s0s2-4343w -p policy/policy_multi_CM0_CM4_tfm.json create-keys
```

Re-provision the new keys to the board:

```
cysecuretools -t cy8ckit-064s0s2-4343w -p policy/policy_multi_CM0_CM4_tfm.json re-provision-device
```

Signing the images

Sign the images using CySecureTools CLI tool. Note: the tool overwrites unsigned file with a signed one, it also creates an unsigned copy <filename>_unsigned.hex.

SPE image:

```
cysecuretools \
--policy platform/ext/target/cypress/psoc64/security/policy/policy_multi_CM0_CM4_tfm.json \
--target cy8ckit-064s0s2-4343w \
sign-image \
--hex <build folder>/bin/tfm_s.hex \
--image-type BOOT \
--image-id 1
```

NSPE image:

```
cysecuretools \
--policy platform/ext/target/cypress/psoc64/security/policy/policy_multi_CM0_CM4_tfm.json \
--target cy8ckit-064s0s2-4343w \
sign-image \
--hex <build folder>/bin/tfm_ns.hex \
--image-type BOOT \
--image-id 16
```

- CySecureTools sign-image overwrites unsigned file with a signed one, also it creates an unsigned copy _unsigned.hex.
- image-type option: "–image-type BOOT" creates a signed hex file with offsets for the primary image slot. Use "–image-type UPGRADE" if you want to create an image for the secondary "upgrade" slot. When booting, CyBootloader will validate image in the secondary slot and copy it to the primary boot slot.
- image-id option: Each image has its own ID. By default, SPE image running on CM0P core has ID=1, NSPE image running on CM4 core has ID=16. Refer to the policy file for the actual ID's.

Programming the Device

After building and signing, the TFM images must be programmed into flash memory on the PSoC64 device. There are three methods to program it.

DAPLink mode

Using KitProg3 mode button, switch it to DAPLink mode. Mode LED should start blinking rapidly and depending on the host computer settings DAPLINK will be mounted as a media storage device. Otherwise, mount it manually.

Copy tfm hex files one by one to the DAPLINK device:

```
cp <build folder>/bin/tfm_ns.hex <mount point>/DAPLINK/; sync
cp <build folder>/bin/tfm_s.hex <mount point>/DAPLINK/; sync
```

OpenOCD

Using KitProg3 mode button, switch to KitProg3 CMSIS-DAP BULK. Status LED should be ON and not blinking for CMSIS-DAP BULK.

Before programming the images, erase PS partition if needed. This will clear all data and force PS to reformat partition.

```
${OPENOCD_PATH}/bin/openocd \
    -s ${OPENOCD_PATH}/scripts \
    -f interface/kitprog3.cfg \
    -f target/psoc6_2m_secure.cfg \
    -c "init; reset init" \
    -c "flash erase_address 0x101c0000 0x10000" \
    -c "shutdown"
```

Note that the 0x101C0000 in the command above must match the PS start address of the secure primary image specified in the file:

platform/ext/target/cypress/psoc64/partition/flash_layout.h

so be sure to change it if you change that file.

To program the signed tfm_s and tfm_ns images to the device with openocd (assuming OPENOCD_PATH is pointing at the openocd installation directory) run the following commands:

```
OPENOCD_PATH=<cyprogrammer dir>/openocd
BUILD_DIR=<build folder>

${OPENOCD_PATH}/bin/openocd \
    -s ${OPENOCD_PATH}/scripts \
    -f interface/kitprog3.cfg \
    -f target/psoc6_2m_secure.cfg \
    -c "init; reset init" \
    -c "flash write_image erase ${BUILD_DIR}/bin/tfm_s.hex" \
    -c "shutdown"

${OPENOCD_PATH}/bin/openocd \
    -s ${OPENOCD_PATH}/scripts \
    -f interface/kitprog3.cfg \
    -f target/psoc6_2m_secure.cfg \
    -c "init; reset init" \
    -c "flash write_image erase ${BUILD_DIR}/bin/tfm_ns.hex" \
    -c "reset run"
```

PyOCD

PyOCD is installed by CySecureTools automatically. It can be used to program TFM images into the board.

Using KitProg3 mode button, switch to KitProg3 DAPLink mode. Mode LED should start blinking rapidly.

Optionally, erase PS partition:

```
pyocd erase -b CY8CKIT-064S0S2-4343W -s 0x101c0000+0x10000
```

To program the signed tfm_s and tfm_ns images to the device with pyocd run the following commands:

```
pyocd flash -b CY8CKIT-064S0S2-4343W ${BUILD_DIR}/bin/tfm_s.hex

pyocd flash -b CY8CKIT-064S0S2-4343W ${BUILD_DIR}/bin/tfm_ns.hex
```

Provisioning device certificates

1. If not done yet, change to the psoc64 security directory and initialize cysecuretools environment:

```
cd platform/ext/target/cypress/psoc64/security
cysecuretools -t cy8ckit-064s0s2-4343w init
```

2. Create and copy rootCA files to “certificates” directory next to the policy directory (please refer to documentation of the used OS)
3. Switch the board in DAPLink mode by pressing the mode button or by issuing the following fw-loader command (fw-loader comes with Modus ToolBox software). Mode LED should be slowly blinking:

```
fw-loader --mode kp3-daplink
```

4. Run reprov_helper.py. If running the script with default parameters, the script can be run as is:

```
 python3 reprov_helper.py

Otherwise, run it with --help parameter to get the full list of options.
```

5. Confirm selected options. When prompted for a serial number, enter the board unique serial number (digits only, e.g. 00183).
6. Script will ask if you want to create new signing keys. Answer Yes to generate new signing keys in the keys directory, or No to retain and use the existing keys. After re-provisioning, from now on any images for this board will have to be signed with these keys.
7. The script will erase user images. Then the script will read device public key and create device certificates based on the board serial number, root certificate and the device public key.

Provisioning Amazon Web Services root and device certificates

Device running Amazon FreeRTOS and using Amazon Web Services (AWS) requires AWS root and unique device certificate signed with the device key for authentication. These certificates have to be provisioned into device. The process is the following:

1. If not done yet, change to the psoc64 security directory and initialize cysecuretools environment:

```
cd platform/ext/target/cypress/psoc64/security
cysecuretools -t cy8ckit-064s0s2-4343w init
```

2. Obtain and copy rootCA files to “certificates” directory next to the policy directory (please refer to AWS documentation)
3. Switch the board in DAPLink mode by pressing the mode button or by issuing the following fw-loader command (fw-loader comes with Modus ToolBox software). Mode LED should be slowly blinking:

```
fw-loader --mode kp3-daplink
```

4. Run reprov_helper.py. If running the script with default parameters, the script can be run as is:

```
 python3 reprov_helper.py

Otherwise, run it with --help parameter to get the full list of options.
```

5. Confirm selected options. When prompted for a serial number, enter the board unique serial number (digits only, e.g. 00183).
6. Script will ask if you want to create new signing keys. Answer Yes to generate new signing keys in the keys directory, or No to retain and use the existing keys. After re-provisioning, from now on any images for this board will have to be signed with these keys.

7. The script will erase user images. Then the script will read device public key and create device certificates based on the board serial number, root certificate and the device public key.

Copyright (c) 2017-2021, Arm Limited. All rights reserved.

Copyright (c) 2019-2020, Cypress Semiconductor Corporation. All rights reserved.

Copyright (c) 2022, Arm Limited. All rights reserved.

Laird Connectivity platform

Laird Connectivity BL5340

The BL5340 development kit (DVK, 750-03293) is a single-board development kit for the evaluation and development on the Laird Connectivity BL5340 module which contains a Nordic Semiconductor nRF5340 System-on-Chip (SoC).

The BL5340 is a module with a dual-core nRF5340 SoC based on the Arm® Cortex®-M33 architecture, with:

- a full-featured ARM Cortex-M33F core with DSP instructions, FPU, and ARMv8-M Security Extension, running at up to 128 MHz, referred to as the **Application MCU**
- a secondary ARM Cortex-M33 core, with a reduced feature set, running at a fixed 64 MHz, referred to as the **Network MCU**.

The BL5340 Application MCU supports the Armv8m Security Extension.

Documentation

The following links provide useful information about the BL5340

BL5340 website:
https://www.lairdconnect.com/wireless-modules/bluetooth-modules/bluetooth-5-modules/bl5340-series-multi-core-bluetooth-52-802154-nfc-modules

Nordic Semiconductor Infocenter: https://infocenter.nordicsemi.com

Building TF-M on BL5340 Application MCU

To build an S and NS application image for the BL5340 Application MCU run the following commands:

Note: On OS X change `readlink` to `greadlink`, available by running `brew install coreutils`.

```
$ mkdir build && cd build
$ cmake -DTFM_PLATFORM=lairdconnectivity/bl5340_dvk_cpuapp \
      -DTFM_TOOLCHAIN_FILE=../toolchain_GNUARM.cmake \
      -G"Unix Makefiles" ../
$ make install
```

Note: Currently, applications can only be built using GCC (GNU ARM Embedded toolchain).

Note: For BL2 (MCUBoot) logging output to be available, the project needs to be built with Debug configuration (CMAKE_BUILD_TYPE=Debug).

Flashing and debugging with on-board Segger J-Link

The BL5340 DVK is equipped with a Debug IC (Atmel ATSAM3U2C) which provides the following functionality:

- Segger J-Link firmware and desktop tools
- SWD debug for the BL5340 module
- USB CDC ACM Serial Port bridged to the second UART peripheral
- Segger RTT Console

To install the J-Link Software and documentation pack, follow the steps below:

#. Download the appropriate package from the J-Link Software and documentation pack website #. Depending on your platform, install the package or run the installer #. When connecting a J-Link-enabled board such as a BL5340 DVK, a serial port should come up

nRF Command-Line Tools Installation

The nRF Command-line Tools allow you to control your BL5340 module from the command line, including resetting it, erasing or programming the flash memory and more.

To install them, visit nRF Command-Line Tools and select your operating system.

After installing, make sure that nrfjprog is somewhere in your executable path to be able to invoke it from anywhere.

BL2, S, and NS application images can be flashed into BL5340 separately or may be merged together into a single binary.

Flashing the BL5340 DVK

To program the flash with a compiled TF-M image (i.e. S, NS or both) after having followed the instructions to install the Segger J-Link Software and the nRF Command-Line Tools, follow the steps below:

Generate Intel hex files from the output binary (bin) files as follows:

```
srec_cat install/outputs/LAIRDCONNECTIVITY/BL5340_DVK_CPUAPP/tfm_s_ns_signed.bin -binary --
↪offset=0x10000 -o install/outputs/LAIRDCONNECTIVITY/BL5340_DVK_CPUAPP/tfm_s_ns_signed.
↪hex -intel
```

- Connect the micro-USB cable to the BL5340 DVK and to your computer
- Erase the flash memory in the BL5340 module:

```
nrfjprog --eraseall -f nrf53
```

- Flash the BL2 and the TF-M image binaries from the sample folder of your choice:

```
nrfjprog --program <sample folder>/install/outputs/LAIRDCONNECTIVITY/BL5340_DVK_CPUAPP/bl2.
↪hex -f nrf53 --sectorerase
nrfjprog --program <sample folder>/install/outputs/LAIRDCONNECTIVITY/BL5340_DVK_CPUAPP/
↪tfm_s_ns_signed.hex -f nrf53 --sectorerase
```

- Reset and start TF-M:

```
nrfjprog --reset -f nrf53
```

Flashing the BL5340 DVK (Secondary slot in QSPI, with BL2)

To program the flash with a compiled TF-M image (i.e. S, NS or both) after having followed the instructions to install the Segger J-Link Software and the nRF Command-Line Tools to the secondary , follow the steps below:

Generate Intel hex files from the output binary (bin) files as follows:

```
srec_cat install/outputs/LAIRDCONNECTIVITY/BL5340_DVK_CPUAPP/tfm_s_ns_signed.bin -binary --
↪offset=0x10000000 -o install/outputs/LAIRDCONNECTIVITY/BL5340_DVK_CPUAPP/tfm_s_ns_
↪signed_qspi.hex -intel
```

- Connect the micro-USB cable to the BL5340 DVK and to your computer
- Erase the flash memory in the BL5340 module:

```
nrfjprog --eraseall -f nrf53
```

- Flash the BL2 and the TF-M image binaries from the sample folder of your choice:

```
nrfjprog --program <sample folder>/install/outputs/LAIRDCONNECTIVITY/BL5340_DVK_CPUAPP/bl2.
↪hex -f nrf53 --sectorerase
nrfjprog --program <sample folder>/install/outputs/LAIRDCONNECTIVITY/BL5340_DVK_CPUAPP/
↪tfm_s_ns_signed.hex -f nrf53 --qspisectorerase
```

- Reset and start TF-M:

```
nrfjprog --reset -f nrf53
```

Secure UART Console on BL5340 DVK

SECURE_UART1 is enabled by default when building TF-M on the BL5340 DVK, so the secure firmware console output is available via USART1.

Non-Secure console output is available via USART0.

Note: By default USART0 and USART1 outputs are routed to separate serial ports.

Copyright (c) 2020, Nordic Semiconductor. All rights reserved. Copyright (c) 2021, Laird Connectivity. All rights reserved.

Nordic platforms

Nordic nRF5340

The nRF5340 development kit (DK) is a single-board development kit for the evaluation and development on the Nordic nRF5340 System-on-Chip (SoC).

The nRF5340 is a dual-core SoC based on the Arm® Cortex®-M33 architecture, with:

- a full-featured ARM Cortex-M33F core with DSP instructions, FPU, and ARMv8-M Security Extension, running at up to 128 MHz, referred to as the **Application MCU**
- a secondary ARM Cortex-M33 core, with a reduced feature set, running at a fixed 64 MHz, referred to as the **Network MCU**.

The nRF5340 Application MCU supports the Armv8m Security Extension.

Documentation

The following links provide useful information about the nRF5340

nRF5340 DK website:
https://www.nordicsemi.com/Software-and-tools/Development-Kits/nRF5340-DK

Nordic Semiconductor Infocenter: https://infocenter.nordicsemi.com

Building TF-M on nRF5340 Application MCU

To build an S and NS application image for the nRF5340 Application MCU run the following commands:

```
cmake -S <TF-M base folder> -B build_spe \
      -DTFM_PLATFORM=nordic_nrf/nrf5340dk_nrf5340_cpuapp \
      -DTFM_TOOLCHAIN_FILE=../toolchain_GNUARM.cmake \
cmake --build build_spe -- install

cmake -S <Application base folder> -B build_app \
      -DCONFIG_SPE_PATH<Absolute path to TF-M base folder>/build_spe/api_ns
      -DTFM_TOOLCHAIN_FILE=cmake/toolchain_ns_GNUARM.cmake
cmake --build
```

Note: Currently, applications can only be built using GCC (GNU ARM Embedded toolchain).

Note: BL2 (MCUBoot) logging output to be available, the project needs to be built with Debug configuration (CMAKE_BUILD_TYPE=Debug).

Flashing and debugging with Nordic nRF Segger J-Link

nRF5340 DK is equipped with a Debug IC (Atmel ATSAM3U2C) which provides the following functionality:

- Segger J-Link firmware and desktop tools
- SWD debug for the nRF5340 IC
- Mass Storage device for drag-and-drop image flashing
- USB CDC ACM Serial Port bridged to the nRFx UART peripheral
- Segger RTT Console
- Segger Ozone Debugger

To install the J-Link Software and documentation pack, follow the steps below:

1. Download the appropriate package from the J-Link Software and documentation pack website
2. Depending on your platform, install the package or run the installer
3. When connecting a J-Link-enabled board such as an nRF5340 DK, a drive corresponding to a USB Mass Storage device as well as a serial port should come up

nRF Command-Line Tools Installation

The nRF Command-line Tools allow you to control your nRF5340 device from the command line, including resetting it, erasing or programming the flash memory and more.

To install them, visit nRF Command-Line Tools and select your operating system.

After installing, make sure that `nrfjprog` is somewhere in your executable path to be able to invoke it from anywhere.

BL2, S, and NS application images can be flashed into nRF5340 separately or may be merged together into a single binary.

Flashing the nRF5340 DK

To program the flash with a compiled TF-M image (i.e. S, NS or both) after having followed the instructions to install the Segger J-Link Software and the nRF Command-Line Tools, follow the steps below:

Generate Intel hex files from the output binary (bin) files as follows:

```
srec_cat build_app/tfm_s_ns_signed.bin -binary --offset 0x10000 \
    -o build_app/tfm_s_ns_signed.hex -intel
```

- Connect the micro-USB cable to the nRF5340 DK and to your computer
- Erase the flash memory in the nRF5340 IC:

```
nrfjprog --eraseall -f nrf53
```

- **(Optionally) Erase the flash memory and reset flash protection and disable** the read back protection mechanism if enabled.

```
nrfjprog --recover -f nrf53
```

- Flash the BL2 and the TF-M image binaries from the sample folder of your choice:

```
nrfjprog --program build_spe/bin/bl2.hex -f nrf53 --sectorerase
nrfjprog --program build_app/tfm_s_ns_signed.hex -f nrf53 --sectorerase
```

- Reset and start TF-M:

```
nrfjprog --reset -f nrf53
```

Secure UART Console on nRF5340 DK

SECURE_UART1 is enabled by default when building TF-M on nRF5340 DK, so the secure firmware console output is available via USART1. When using the nRF5340 network MCU make sure to configure different UART pins or disable SECURE_UART1.

Non-Secure console output is available via USART0.

Note: By default USART0 and USART1 outputs are routed to separate serial ports.

Copyright (c) 2020, Nordic Semiconductor. All rights reserved.

Nordic nRF9160

The nRF9160 development kit (DK) is a single-board development kit for the evaluation and development on the Nordic nRF9160 SiP for LTE-M and NB-IoT.

The nRF9160 SoC features a full-featured Arm® Cortex®-M33F core with DSP instructions, FPU, and ARMv8-M Security Extension, running at up to 64 MHz.

Documentation

The following links provide useful information about the nRF9160

nRF9160 DK website:
https://www.nordicsemi.com/Software-and-tools/Development-Kits/nRF9160-DK

Nordic Semiconductor Infocenter: https://infocenter.nordicsemi.com

Building TF-M on nRF9160

To build an S and NS application image for the nRF9160 run the following commands:

```
cmake -S <TF-M base folder> -B build_spe \
      -DTFM_PLATFORM=nordic_nrf/nrf9160dk_nrf9160 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_GNUARM.cmake \
cmake --build build_spe -- install

cmake -S <Application base folder> -B build_app \
      -DCONFIG_SPE_PATH<Absolute path to TF-M base folder>/build_spe/api_ns
      -DTFM_TOOLCHAIN_FILE=cmake/toolchain_ns_GNUARM.cmake
cmake --build
```

Note: Currently, applications can only be built using GCC (GNU ARM Embedded toolchain).

Note: For BL2 (MCUBoot) logging output to be available, the project needs to be built with Debug configuration (CMAKE_BUILD_TYPE=Debug).

Flashing and debugging with Nordic nRF Segger J-Link

nRF9160 DK is equipped with a Debug IC (Atmel ATSAM3U2C) which provides the following functionality:

- Segger J-Link firmware and desktop tools
- SWD debug for the nRF9160 IC
- Mass Storage device for drag-and-drop image flashing
- USB CDC ACM Serial Port bridged to the nRFx UART peripheral
- Segger RTT Console
- Segger Ozone Debugger

To install the J-Link Software and documentation pack, follow the steps below:

1. Download the appropriate package from the J-Link Software and documentation pack website
2. Depending on your platform, install the package or run the installer
3. When connecting a J-Link-enabled board such as an nRF9160 DK, a drive corresponding to a USB Mass Storage device as well as a serial port should come up

nRFx Command-Line Tools Installation

The nRF Command-line Tools allow you to control your nRF9160 device from the command line, including resetting it, erasing or programming the flash memory and more.

To install them, visit nRF Command-Line Tools and select your operating system.

After installing, make sure that nrfjprog is somewhere in your executable path to be able to invoke it from anywhere.

BL2, S, and NS application images can be flashed into nRF9160 separately or may be merged together into a single binary.

Flashing the nRF9160 DK

To program the flash with a compiled TF-M image (i.e. S, NS or both) after having followed the instructions to install the Segger J-Link Software and the nRFx Command-Line Tools, follow the steps below:

Generate Intel hex files from the output binary (bin) files as follows:

```
srec_cat build_app/tfm_s_ns_signed.bin -binary --offset 0x10000 \
    -o build_app/tfm_s_ns_signed.hex -intel
```

- Connect the micro-USB cable to the nRF9160 DK and to your computer
- Erase the flash memory in the nRF9160 IC:

```
nrfjprog --eraseall -f nrf91
```

- **(Optionally) Erase the flash memory and reset flash protection and disable** the read back protection mechanism if enabled.

```
nrfjprog --recover -f nrf91
```

- Flash the BL2 and TF-M image binaries from the sample folder of your choice:

```
nrfjprog --program build_spe/bin/bl2.hex -f nrf91 --sectorerase
nrfjprog --program build_app/tfm_s_ns_signed.hex -f nrf91 --sectorerase
```

- Reset and start TF-M:

```
nrfjprog --reset -f nrf91
```

Secure UART Console on nRF9160 DK

SECURE_UART1 is enabled by default when building TF-M on nRF9160 DK, so the secure firmware console output is available via USART1.

Non-Secure console output is available via USART0.

Note: By default USART0 and USART1 outputs are routed to separate serial ports.

Copyright (c) 2020, Arm Limited. All rights reserved. Copyright (c) 2020, Nordic Semiconductor. All rights reserved.

Nordic nRF9161

The nRF9161 development kit (DK) is a single-board development kit for the evaluation and development on the Nordic nRF9161 SiP for LTE-M and NB-IoT.

The nRF9161 SoC features a full-featured Arm® Cortex®-M33F core with DSP instructions, FPU, and ARMv8-M Security Extension, running at up to 64 MHz.

Documentation

The following links provide useful information about the nRF9161

nRF9161 DK website:
https://www.nordicsemi.com/products/nrf9161

Nordic Semiconductor Infocenter: https://infocenter.nordicsemi.com

Building TF-M on nRF9161

To build an S and NS application image for the nRF9161 run the following commands:

```
cmake -S <TF-M base folder> -B build_spe \
      -DTFM_PLATFORM=nordic_nrf/nrf9161dk_nrf9161 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_GNUARM.cmake \
cmake --build build_spe -- install

cmake -S <Application base folder> -B build_app \
      -DCONFIG_SPE_PATH<Absolute path to TF-M base folder>/build_spe/api_ns
      -DTFM_TOOLCHAIN_FILE=cmake/toolchain_ns_GNUARM.cmake
cmake --build
```

Note: Currently, applications can only be built using GCC (GNU ARM Embedded toolchain).

Note: For BL2 (MCUBoot) logging output to be available, the project needs to be built with Debug configuration (CMAKE_BUILD_TYPE=Debug).

Flashing and debugging with Nordic nRF Segger J-Link

nRF9161 DK is equipped with a Debug IC (Atmel ATSAM3U2C) which provides the following functionality:

- Segger J-Link firmware and desktop tools
- SWD debug for the nRF9161 IC
- Mass Storage device for drag-and-drop image flashing
- USB CDC ACM Serial Port bridged to the nRFx UART peripheral
- Segger RTT Console
- Segger Ozone Debugger

To install the J-Link Software and documentation pack, follow the steps below:

1. Download the appropriate package from the J-Link Software and documentation pack website
2. Depending on your platform, install the package or run the installer
3. When connecting a J-Link-enabled board such as an nRF9161 DK, a drive corresponding to a USB Mass Storage device as well as a serial port should come up

nRFx Command-Line Tools Installation

The nRF Command-line Tools allow you to control your nRF9161 device from the command line, including resetting it, erasing or programming the flash memory and more.

To install them, visit nRF Command-Line Tools and select your operating system.

After installing, make sure that `nrfjprog` is somewhere in your executable path to be able to invoke it from anywhere.

BL2, S, and NS application images can be flashed into nRF9161 separately or may be merged together into a single binary.

Flashing the nRF9161 DK

To program the flash with a compiled TF-M image (i.e. S, NS or both) after having followed the instructions to install the Segger J-Link Software and the nRFx Command-Line Tools, follow the steps below:

Generate Intel hex files from the output binary (bin) files as follows:

```
srec_cat build_app/tfm_s_ns_signed.bin -binary --offset 0x10000 \
    -o build_app/tfm_s_ns_signed.hex -intel
```

- Connect the micro-USB cable to the nRF9161 DK and to your computer
- Erase the flash memory in the nRF9161 IC:

```
nrfjprog --eraseall -f nrf91
```

- **(Optionally) Erase the flash memory and reset flash protection and disable** the read back protection mechanism if enabled.

```
nrfjprog --recover -f nrf91
```

- Flash the BL2 and TF-M image binaries from the sample folder of your choice:

```
nrfjprog --program build_spe/bin/bl2.hex -f nrf91 --sectorerase
nrfjprog --program build_app/tfm_s_ns_signed.hex -f nrf91 --sectorerase
```

- Reset and start TF-M:

```
nrfjprog --reset -f nrf91
```

Secure UART Console on nRF9161 DK

SECURE_UART1 is enabled by default when building TF-M on nRF9161 DK, so the secure firmware console output is available via USART1.

Non-Secure console output is available via USART0.

Note: By default USART0 and USART1 outputs are routed to separate serial ports.

Copyright (c) 2023, Nordic Semiconductor. All rights reserved.

Nuvoton platforms

NuMaker-PFM-M2351

Building TF-M

Build TF-M regression test with M2351 by following commands:

```
$ cd tf-m-test/tests_reg
$ cmake -S spe -B build_spe \
        -G"Unix Makefiles" \
        -DTFM_PLATFORM=nuvoton/m2351 \
        -DTFM_TOOLCHAIN_FILE=[tf-m path]/toolchain_GNUARM.cmake \
        -DCMAKE_BUILD_TYPE=Release \
        -DTEST_S=ON -DTEST_NS=ON
$ cmake --build build --parallel -- install
$ cmake -S . -B build_test \
        -G"Unix Makefiles" \
        -DCONFIG_SPE_PATH=[tf-m-tests path]/tests_reg/build_spe/api_ns \
        -DTFM_TOOLCHAIN_FILE=cmake/toolchain_ns_GNUARM.cmake \
        -DCMAKE_BUILD_TYPE=Release \
$ cmake --build build_test --parallel -- install
```

Define TEST_NS=ON or TEST_S=ON for non-secure or secure regression test. CMAKE_BUILD_TYPE could be “Release”, “RelWithDebInfo” or “Minsizerel” Other cmake parameters should not be changed.

Flashing Image with Nuvoton NuLink Tool

NuMaker-PFM-M2351 board supports on board ICE called NuLink. Connect NuLink to PC with USB cable and using NuLink Command Tool to flashing the image.

The commands are as follows:

```
> NuLink_M2351_M261 -C
> NuLink_M2351_M261 -E ALL
> NuLink_M2351_M261 -W NSCBA 0x80000 0
> NuLink_M2351_M261 -S
> NuLink_M2351_M261 -C
> NuLink_M2351_M261 -W APROM .\build_spe\bin\bl2.bin 0
> NuLink_M2351_M261 -W APROM .\build_spe\bin\tfm_s_signed.bin 0 0x20000 0
> NuLink_M2351_M261 -W APROMNS .\build_test\bin\tfm_ns_signed.bin 0
```

Copyright (c) 2021-2023, Nuvoton Technology Corp. All rights reserved.

NuMaker-PFM-M2354

Building TF-M

Build TF-M regression test with M2354 by following commands:

```
$ cd tf-m-test/tests_reg
$ cmake -S spe -B build_spe \
      -G"Unix Makefiles" \
      -DTFM_PLATFORM=nuvoton/m2354 \
      -DTFM_TOOLCHAIN_FILE=[tf-m path]/toolchain_GNUARM.cmake \
      -DCMAKE_BUILD_TYPE=Release \
      -DTEST_S=ON -DTEST_NS=ON
$ cmake --build build --parallel -- install
$ cmake -S . -B build_test \
      -G"Unix Makefiles" \
      -DCONFIG_SPE_PATH=[tf-m-tests path]/tests_reg/build_spe/api_ns \
      -DTFM_TOOLCHAIN_FILE=cmake/toolchain_ns_GNUARM.cmake \
      -DCMAKE_BUILD_TYPE=Release \
$ cmake --build build_test --parallel -- install
```

Define TEST_NS=ON or TEST_S=ON for non-secure or secure regression test. CMAKE_BUILD_TYPE could be “Release”, “RelWithDebInfo” or “Minsizerel” Other cmake parameters should not be changed.

Flashing Image with Nuvoton NuLink Tool

NuMaker-PFM-M2354 board supports on board ICE called NuLink. Connect NuLink to PC with USB cable and using NuLink Command Tool to flashing the image.

The commands are as follows:

```
> NuLink_M2354 -C
> NuLink_M2354 -E ALL
> NuLink_M2354 -W NSCBA 0x80000 0
> NuLink_M2354 -S
> NuLink_M2354 -C
> NuLink_M2354 -W APROM .\build_spe\bin\bl2.bin 0
```

(continues on next page)

(continued from previous page)

```
> NuLink_M2354 -W APROM .\build_spe\bin\tfm_s_signed.bin 0 0x20000 0
> NuLink_M2354 -W APROMNS .\build_test\bin\tfm_ns_signed.bin 0
```

Copyright (c) 2021-2023, Nuvoton Technology Corp. All rights reserved.

NXP platform

LPCXpresso55S69

1. Building TF-M

Prepare the tf-m-tests repository inside the TF-M base folder.

```
cd <TF-M base folder>
git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
```

There are two options for the TF-M build - with or without secondary bootloader (BL2).

1.1 Building TF-M demo without BL2

To build S and NS application image without a BL2, run the following commands:

```
cd <TF-M base folder>/tf-m-test/tests_reg
cmake -S spe -B build_spe -DTFM_PLATFORM=nxp/lpcxpresso55s69 -DCONFIG_TFM_SOURCE_
↪PATH=<TF-M base folder>/trusted-firmware-m -G"Unix Makefiles" -DBL2=OFF
cmake --build build_spe -- install
cmake -S . -B build_test -G"Unix Makefiles" -DCONFIG_SPE_PATH=<TF-M base folder>/tf-m-tests/
↪tests_reg/build_spe/api_ns
cmake --build build_test
```

1.2 Building TF-M demo with BL2

To build S and NS application along with a BL2, run the following commands:

```
cd <TF-M base folder>/tf-m-test/tests_reg
cmake -S spe -B build_spe -DTFM_PLATFORM=nxp/lpcxpresso55s69 -DCONFIG_TFM_SOURCE_
↪PATH=<TF-M base folder>/trusted-firmware-m -G"Unix Makefiles" -DBL2=ON
cmake --build build_spe -- install
cmake -S . -B build_test -G"Unix Makefiles" -DCONFIG_SPE_PATH=<TF-M base folder>/tf-m-tests/
↪tests_reg/build_spe/api_ns
cmake --build build_test
```

1.3 Building TF-M regression tests

To buld the S and NS regression tests without BL2, run the following commands:

```
cd <TF-M base folder>/tf-m-test/tests_reg
cmake -S spe -B build_spe -DTFM_PLATFORM=nxp/lpcxpresso55s69 -DCONFIG_TFM_SOURCE_
↪PATH=<TF-M base folder>/trusted-firmware-m -G"Unix Makefiles" -DTEST_S=ON -DTEST_NS=ON -
↪DBL2=OFF
cmake --build build_spe -- install
cmake -S . -B build_test -G"Unix Makefiles" -DCONFIG_SPE_PATH=<TF-M base folder>/tf-m-tests/
↪tests_reg/build_spe/api_ns
cmake --build build_test
```

2. Flashing TF-M with PyOCD

PyOCD is an open source Python package for programming and debugging Arm Cortex-M microcontrollers using multiple supported types of USB debug probes. See: *PyOCD <https://pypi.org/project/pyocd/>*

If you built TF-M with the BL2 secondary bootloader, use the following commands:

```
pyocd erase --mass -t LPC55S69
pyocd flash build_spe/bin/bl2.hex -t LPC55S69
pyocd flash build_test/tfm_s_ns_signed.bin --base-address 0x8000 -t LPC55S69
```

When BL2 is disabled, flash the generated hex secure and non-secure images:

```
pyocd erase --mass -t LPC55S69
pyocd flash build_spe/bin/tfm_s.hex build_test/bin/tfm_ns.hex -t LPC55S69
```

Note: There is sometimes a stability issue with the pyocd flash erasing, it is needed to rerun the pyocd erase command again.

---

*Copyright (c) 2020, Linaro. All rights reserved. Copyright (c) 2020-2021, Arm Limited. All rights reserved. Copyright 2020-2024 NXP. All rights reserved. *SPDX-License-Identifier: BSD-3-Clause*

Raspberry Pi platforms

RP2350

Introduction

RP2350 features a dual-core Arm Cortex-M33 processor with 520 kiB on-chip SRAM, support for up to 16MB of off-chip flash and a wide range of flexible I/O option including I2C, SPI, and - uniquely - Programmable I/O (PIO). With its security features RP2350 offers significant enhancements over RP2040.

This platform port supports TF-M regression tests (Secure and Non-secure) with Isolation Level 1 and 2.

Note: Only GNU toolchain is supported.

Note: Only “profile_medium” predefined profile is supported.

Building TF-M

Follow the instructions in Building instructions.

Build instructions with platform name: rpi/rp2350

-DTFM_PLATFORM=rpi/rp2350

Note: This platform port relies on Raspberry Pi Pico SDK. Make sure it is either cloned locally or available to download during build. SDK version used for testing: SDK 2.1.1 release.

To use a different version of the SDK than the one your version of TF-M uses by default, you can pass -DPICO_SDK_PATH=/path/to/pico-sdk to CMake. You may also need to patch this SDK to work with the TF-M build system. If required, the patch is available in the platform/ext/target/rpi/rp2350/pico-sdk.patch file from a TF-M version that uses the SDK version you want to use, and it can be applied with git apply <TF-M source dir>/platform/ext/target/rpi/rp2350/pico-sdk.patch.

For example, to use SDK version 2.1.1, you can take the pico-sdk.patch file from TF-M version 2.1.2 and apply it to the SDK version 2.1.1.

Note: Building the default platform configuration requires the board to be provisioned. For this the provision bundle needs to be built and run on the board with -DPLATFORM_DEFAULT_PROVISIONING=OFF. The binary must be placed in flash at the address defined by PROVISIONING_BUNDLE_START. One way to do this is to generate a .uf2 file containing the bundle at the start address and copy it to the board. There is an example in the provided pico_uf2.sh script and in the description below.

If -DPLATFORM_DEFAULT_PROVISIONING=OFF and -DTFM_DUMMY_PROVISIONING=ON then the keys in the <TF-M source dir>/platform/ext/common/provisioning_bundle/provisioning_config.cmake and the default MCUBoot signing keys will be used for provisioning.

If -DPLATFORM_DEFAULT_PROVISIONING=OFF and -DTFM_DUMMY_PROVISIONING=OFF are set then unique values can be used for provisioning. The keys and seeds can be changed by passing the new values to the build command, or by setting the -DPROVISIONING_KEYS_CONFIG flag to a .cmake file that contains the keys. An example config cmake file can be seen at <TF-M source dir>/platform/ext/common/provisioning_bundle/provisioning_config.cmake. Otherwise new random values are going to be generated and used. For the image signing the ${MCUBOOT_KEY_S} and ${MCUBOOT_KEY_NS} will be used. These variables should point to .pem files that contain the code signing private keys. The public keys are going to be generated from these private keys and will be used for provisioning. The hash of the public key is going to be written into the provisioning_data.c automatically.

If -DMCUBOOT_GENERATE_SIGNING_KEYPAIR=ON is set then a new mcuboot signing public and private keypair is going to be generated and it's going to be used to sign the S and NS binaries.

The new generated keypair can be found in the <build dir>/bin folder or in the <install directory>/image_signing/keys after installation. The generated provisioning_data.c file can be found at <build dir>/platform/target/provisioning/provisioning_data.c

---

Note: The provisioning bundle generation depends on pyelftools that needs to be installed via:

```
pip3 install pyelftools
```

Example of build instructions for regression tests with dummy keys:

Building Secure side with provisioning bundle:

Note: Add optionally:

- -DTFM_MULTI_CORE_TOPOLOGY=ON for multicore support
- -DPICO_SDK_PATH=<abs-path-to-pico-sdk> for a pre-fetched Pico SDK

```
cmake -S <TF-M-tests source dir>/tests_reg/spe \
-B <TF-M-tests source dir>/tests_reg/spe/build_rpi_single \
-DTFM_PLATFORM=rpi/rp2350 \
```

(continues on next page)

(continued from previous page)

```
-DTFM_TOOLCHAIN_FILE=<TF-M source dir>/toolchain_GNUARM.cmake \
-DCONFIG_TFM_SOURCE_PATH=<TF-M source dir> \
-DTFM_PROFILE=profile_medium \
-DPLATFORM_DEFAULT_PROVISIONING=OFF \
-DTEST_S=ON \
-DTEST_NS=ON
```

```
cmake --build <TF-M-tests source dir>/tests_reg/spe/build_rpi_single \
-- -j8 install
```

Building Non-Secure side:

```
cmake -S <TF-M-tests source dir>/tests_reg \
-B <TF-M-tests source dir>/tests_reg/build_rpi_single \
-DCONFIG_SPE_PATH=<TF-M tests source dir>/tests_reg/spe/build_rpi_single/api_ns \
-DTFM_TOOLCHAIN_FILE=<TF-M-tests source dir>/tests_reg/spe/build_rpi_single/api_ns/cmake/
↪toolchain_ns_GNUARM.cmake
```

```
cmake --build <TF-M-tests source dir>/tests_reg/build_rpi_single -- -j8
```

Binaries need to be converted with a small script pico_uf2.sh. It requires uf2conv.py from here: https://github.com/microsoft/uf2/blob/master/utils/uf2conv.py. It depends on: https://github.com/microsoft/uf2/blob/master/utils/uf2families.json. Both the above files need to be copied into the same place where pico_uf2.sh runs. Also, you may need to give executable permissions to both pico_uf2.sh and uf2conv.py. The tool takes the combined and signed S and NS images in .bin format, and generates the corresponding .uf2 file. It also generates the .uf2 for the bootloader (bl2.uf2) and the provisioning bundle one.

```
pico_uf2.sh <TF-M-tests source dir> build_rpi_single
```

Then just copy (drag-and-drop) the bl2.uf2 and tfm_s_ns_signed.uf2 files into the board, one at time. It will run the BL2, S and NS tests and print the results to the UART (Baudrate 115200). If the board needs provisioning, the .uf2 file containing the provisioning bundle needs to be copied before tfm_s_ns_signed.uf2. It only needs to be done once.

Note: If a different application was copied to the board before, erasing the flash might be necessary.

Erasing the flash:

Generating flash sized image of zeros can be done with the truncate command, then it can be converted to the uf2 format with the uf2conv.py utility. The resulting uf2 file then needs to be copied to the board. Current platform flash size is 2MB, please adjust size based on your board specs (PICO_FLASH_SIZE_BYTES):

```
truncate -s 2M nullbytes2M.bin
uf2conv.py nullbytes2M.bin --base 0x10000000 --convert --output nullbytes2M.uf2 --family 0xe48bff59
```

SPDX-License-Identifier: BSD-3-Clause SPDX-FileCopyrightText: Copyright The TrustedFirmware-M Contributors

---

SPDX-License-Identifier: BSD-3-Clause SPDX-FileCopyrightText: Copyright The TrustedFirmware-M Contributors

STMICROELECTRONICS

This document contains SOC and boards supported by stm32.

STM32H5

TF-M is supported on STM32H5 family

https://www.st.com/en/microcontrollers-microprocessors/stm32h5-series.html

Directory content

- **stm/common/stm32h5xx/stm32h5xx_hal:**
  Content from https://github.com/STMicroelectronics/stm32h5xx_hal_driver
- **stm/common/stm32h5xx/Device:**
  Content from https://github.com/STMicroelectronics/cmsis_device_h5
- **stm/common/stm32h5xx/bl2:**
  stm32h5xx bl2 code specific from https://github.com/STMicroelectronics/STM32CubeH5.git (Projects/STM32H573I_DK/Applications/TFM)
- **stm/common/stm32h5xx/secure:**
  stm32h5xx Secure porting adaptation from https://github.com/STMicroelectronics/STM32CubeH5.git (Projects/STM32H573I_DK/Applications/TFM)
- **stm/common/stm32h5xx/boards:**
  Adaptation and tools specific to stm32 board using stm32h5xx soc from https://github.com/STMicroelectronics/STM32CubeH5.git (Projects/STM32H573I_DK/Applications/TFM)
- **stm/common/stm32h5xx/CMSIS_Driver:**
  Flash and uart driver for stm32h5xx platform
- **stm/common/stm32h5xx/Native_Driver:**
  Random generator and tickless implementation

Specific Software Requirements

STM32_Programmer_CLI is required.(see https://www.st.com/en/development-tools/stm32cubeprog.html)

Limitations to Consider When Using the Platform

MPU and ICACHE disabled in bl2 boot stage

Write software on target

In build folder:

- postbuild.sh: Updates regression.sh and TFM_UPDATE.sh scripts according to flash_layout.h
- regression.sh: Sets platform option bytes config and erase platform
- TFM_UPDATE.sh: Writes bl2, secure, and non secure image in target

Connect board to USB and Execute the 3 scripts in following order to update platform: post-build.sh, regression.sh, TFM_UPDATE.sh

The virtual com port from STLINK is used for TFM log and serial port configuration should be:

- Baud rate = 115200
- Data = 8 bits
- Parity = none
- Stop = 1 bit

- Flow control = none

Copyright (c) 2023 STMicroelectronics. All rights reserved. SPDX-License-Identifier: BSD-3-Clause

STM32L5

TF-M is supported on STM32L5 family

https://www.st.com/en/microcontrollers-microprocessors/stm32l5-series.html

https://www.st.com/resource/en/user_manual/dm00678763-getting-started-with-stm32cubel5-tfm-application-s
pdf

Directory content

- **stm/common/stm32l5xx/stm32l5xx_hal:**
  Content from https://github.com/STMicroelectronics/stm32l5xx_hal_driver.git
- **stm/common/stm32l5xx/Device:**
  Content from https://github.com/STMicroelectronics/cmsis_device_l5.git
- **stm/common/stm32l5xx/bl2:**
  stm32l5xx bl2 code specific from https://github.com/STMicroelectronics/STM32CubeL5.git (Projects/STM32L562E-DK/Applications/TFM)
- **stm/common/stm32l5xx/secure:**
  stm32l5xx Secure porting adaptation from https://github.com/STMicroelectronics/STM32CubeL5.git (Projects/STM32L562E-DK/Applications/TFM)
- **stm/common/stm32l5xx/boards:**
  Adaptation and tools specific to stm32 board using stm32l5xx soc from https://github.com/STMicroelectronics/STM32CubeL5.git (Projects/STM32L562E-DK/Applications/TFM)
- **stm/common/stm32l5xx/CMSIS_Driver:**
  Flash and uart driver for stm32l5xx platform
- **stm/common/stm32l5xx/Native_Driver:**
  Random generator and tickless implementation

Specific Software Requirements

STM32_Programmer_CLI is required.(see https://www.st.com/en/development-tools/stm32cubeprog.html)

Write software on target

In secure build folder directory api_ns:

- postbuild.sh: Updates regression.sh and TFM_UPDATE.sh scripts according to flash_layout.h
- regression.sh: Sets platform option bytes config and erase platform
- TFM_UPDATE.sh: Writes bl2, secure, and non secure image in target

Connect board to USB and Execute the 3 scripts in following order to update platform: post-build.sh, regression.sh, TFM_UPDATE.sh

The virtual com port from STLINK is used for TFM log and serial port configuration should be:

- Baud rate = 115200
- Data = 8 bits
- Parity = none

- Stop = 1 bit
- Flow control = none

Copyright (c) 2019 STMicroelectronics. All rights reserved. SPDX-License-Identifier: BSD-3-Clause

STM32U5

TF-M is supported on STM32U5 family

https://www.st.com/en/microcontrollers-microprocessors/stm32u5-series.html

Directory content

- **stm/common/stm32u5xx/stm32u5xx_hal:**
  Content from https://github.com/STMicroelectronics/stm32u5xx_hal_driver.git
- **stm/common/stm32u5xx/Device:**
  Content from https://github.com/STMicroelectronics/cmsis_device_u5.git
- **stm/common/stm32u5xx/bl2:**
  stm32l5xx bl2 code specific from https://github.com/STMicroelectronics/STM32CubeU5.git (Projects/B-U585I-IOT02A/Applications/TFM)
- **stm/common/stm32u5xx/secure:**
  stm32l5xx Secure porting adaptation from https://github.com/STMicroelectronics/STM32CubeU5.git (Projects/B-U585I-IOT02A/Applications/TFM)
- **stm/common/stm32u5xx/boards:**
  Adaptation for stm32 board using stm32l5xx soc from https://github.com/STMicroelectronics/STM32CubeU5.git (Projects/B-U585I-IOT02A/Applications/TFM)

Specific Software Requirements

STM32_Programmer_CLI is required.(see https://www.st.com/en/development-tools/stm32cubeprog.html)

Write software on target

In secure build folder directory api_ns:

- postbuild.sh: Updates regression.sh and TFM_UPDATE.sh scripts according to flash_layout.h
- regression.sh: Sets platform option bytes config and erase platform
- TFM_UPDATE.sh: Writes bl2, secure, and non secure image in target

Connect board to USB and Execute the 3 scripts in following order to update platform: post-build.sh, regression.sh, TFM_UPDATE.sh

The virtual com port from STLINK is used for TFM log and serial port configuration should be:

- Baud rate = 115200
- Data = 8 bits
- Parity = none
- Stop = 1 bit
- Flow control = none

Copyright (c) 2021 STMicroelectronics. All rights reserved. SPDX-License-Identifier: BSD-3-Clause

NUCLEO_L552ZE_Q

STM32 Nucleo-144 development board with STM32L552ZE MCU, SMPS, supports Arduino, ST Zio and morpho connectivity https://www.st.com/en/evaluation-tools/nucleo-l552ze-q.html

Configuration and Build

GNUARM/ARMCLANG/IARARM compilation is available for this target. and build the selected configuration as follow.

The build configuration for TF-M is provided to the build system using command line arguments. Required arguments are noted below.

The following instructions build multi-core TF-M with regression test suites in Isolation Level 1.

In common STM (platform\ext\target\stm\common\build_stm) There are scripts that help users to build the TF-M project on all STM platforms

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
git checkout <recommended tf-m-tests commit>

mkdir build_s && cd build_s

cmake -S /../tf-m-tests/tests_reg/spe -B . -GNinja -DTFM_PLATFORM=stm/nucleo_l552ze_q
    -DTFM_TOOLCHAIN_FILE= /../toolchain_ARMCLANG.cmake
    -DCONFIG_TFM_SOURCE_PATH= /../trusted-firmware-m
    -DTFM_PSA_API=ON -DTFM_ISOLATION_LEVEL=1
    -DTEST_S=ON -DTEST_NS=ON

ninja -C . install -j 8

cd <trusted-firmware-m folder>
mkdir build_ns && cd build_ns
cmake -S /../trusted-firmware-m  /../tf-m-tests/tests_reg -B . -GNinja
    -DCONFIG_SPE_PATH= /../build_s/api_ns -DTFM_TOOLCHAIN_FILE= /../build_s/api_ns/
↪cmake/toolchain_ns_ARMCLANG.cmake

ninja -C . -j 8
```

The following instructions build multi-core TF-M with PSA API test suite for the attestation service in Isolation Level 1 on Linux.

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
git checkout <recommended tf-m-tests commit>

mkdir build_s && cd build_s

cmake -S /../tf-m-tests/tests_psa_arch/spe -B . -GNinja -DTFM_PLATFORM=stm/nucleo_l552ze_q
    -DTFM_TOOLCHAIN_FILE= /../toolchain_ARMCLANG.cmake
    -DCONFIG_TFM_SOURCE_PATH= /../trusted-firmware-m
    -DTFM_PSA_API=ON -DTFM_ISOLATION_LEVEL=1
    -DTEST_PSA_API=INITIAL_ATTESTATION

ninja -C . install -j 8

cd <trusted-firmware-m folder>
mkdir build_ns && cd build_ns
```

(continues on next page)

(continued from previous page)

```
cmake -S /../trusted-firmware-m  /../tf-m-tests/tests_psa_arch -B . -GNinja
    -DCONFIG_SPE_PATH= /../build_s/api_ns -DTFM_TOOLCHAIN_FILE= /../build_s/api_ns/
↪cmake/toolchain_ns_ARMCLANG.cmake

ninja -C . -j 8
```

Copyright (c) 2021, Arm Limited. All rights reserved. Copyright (c) 2019, STMicroelectronics. All rights reserved. SPDX-License-Identifier: BSD-3-Clause

STM32H573I_DK

Discovery kit for IoT node with STM32H5 series https://www.st.com/en/evaluation-tools/stm32h573i-dk.html

Configuration and Build

GNUARM/ARMCLANG/IARARM compilation is available for this target. and build the selected configuration as follow.

The build configuration for TF-M is provided to the build system using command line arguments. Required arguments are noted below.

The following instructions build multi-core TF-M with regression test suites in Isolation Level 1.

In common STM (platform\ext\target\stm\common\build_stm) There are scripts that help users to build the TF-M project on all STM platforms

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
git checkout <recommended tf-m-tests commit>

mkdir build_s && cd build_s

cmake -S /../tf-m-tests/tests_reg/spe -B . -GNinja -DTFM_PLATFORM=stm/stm32h573i_dk
    -DTFM_TOOLCHAIN_FILE= /../toolchain_ARMCLANG.cmake
    -DCONFIG_TFM_SOURCE_PATH= /../trusted-firmware-m
    -DTFM_PSA_API=ON -DTFM_ISOLATION_LEVEL=1
    -DTEST_S=ON -DTEST_NS=ON

ninja -C . install -j 8

cd <trusted-firmware-m folder>
mkdir build_ns && cd build_ns
cmake -S /../trusted-firmware-m  /../tf-m-tests/tests_reg -B . -GNinja
    -DCONFIG_SPE_PATH= /../build_s/api_ns -DTFM_TOOLCHAIN_FILE= /../build_s/api_ns/
↪cmake/toolchain_ns_ARMCLANG.cmake

ninja -C . -j 8
```

The following instructions build multi-core TF-M with PSA API test suite for the attestation service in Isolation Level 1 on Linux.

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
git checkout <recommended tf-m-tests commit>
```

(continues on next page)

(continued from previous page)

```
mkdir build_s && cd build_s

cmake -S /../tf-m-tests/tests_psa_arch/spe -B . -GNinja -DTFM_PLATFORM=stm/stm32h573i_dk
    -DTFM_TOOLCHAIN_FILE= /../toolchain_ARMCLANG.cmake
    -DCONFIG_TFM_SOURCE_PATH= /../trusted-firmware-m
    -DTFM_PSA_API=ON -DTFM_ISOLATION_LEVEL=1
    -DTEST_PSA_API=INITIAL_ATTESTATION

ninja -C . install -j 8

cd <trusted-firmware-m folder>
mkdir build_ns && cd build_ns
cmake -S /../trusted-firmware-m  /../tf-m-tests/tests_psa_arch -B . -GNinja
    -DCONFIG_SPE_PATH= /../build_s/api_ns -DTFM_TOOLCHAIN_FILE= /../build_s/api_ns/
↪cmake/toolchain_ns_ARMCLANG.cmake

ninja -C . -j 8
```

Copyright (c) 2023, STMicroelectronics. All rights reserved. SPDX-License-Identifier: BSD-3-Clause

STM32L562E_DK

Discovery kit with STM32L562QE MCU : https://www.st.com/en/evaluation-tools/stm32l562e-dk.html

Configuration and Build

GNUARM/ARMCLANG/IARARM compilation is available for this target. and build the selected configuration as follow.

The build configuration for TF-M is provided to the build system using command line arguments. Required arguments are noted below.

The following instructions build multi-core TF-M with regression test suites in Isolation Level 1.

In common STM (platform\ext\target\stm\common\build_stm) There are scripts that help users to build the TF-M project on all STM platforms

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
git checkout <recommended tf-m-tests commit>

mkdir build_s && cd build_s

cmake -S /../tf-m-tests/tests_reg/spe -B . -GNinja -DTFM_PLATFORM=stm/stm32l562e_dk
    -DTFM_TOOLCHAIN_FILE= /../toolchain_ARMCLANG.cmake
    -DCONFIG_TFM_SOURCE_PATH= /../trusted-firmware-m
    -DTFM_PSA_API=ON -DTFM_ISOLATION_LEVEL=1
    -DTEST_S=ON -DTEST_NS=ON

ninja -C . install -j 8

cd <trusted-firmware-m folder>
mkdir build_ns && cd build_ns
cmake -S /../trusted-firmware-m  /../tf-m-tests/tests_reg -B . -GNinja
    -DCONFIG_SPE_PATH= /../build_s/api_ns -DTFM_TOOLCHAIN_FILE= /../build_s/api_ns/
↪cmake/toolchain_ns_ARMCLANG.cmake
```

(continues on next page)

(continued from previous page)

```
ninja -C . -j 8
```

The following instructions build multi-core TF-M with PSA API test suite for the attestation service in Isolation Level 1 on Linux.

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
git checkout <recommended tf-m-tests commit>

mkdir build_s && cd build_s

cmake -S /../tf-m-tests/tests_psa_arch/spe -B . -GNinja -DTFM_PLATFORM=stm/stm32l562e_dk
    -DTFM_TOOLCHAIN_FILE= /../toolchain_ARMCLANG.cmake
    -DCONFIG_TFM_SOURCE_PATH= /../trusted-firmware-m
    -DTFM_PSA_API=ON -DTFM_ISOLATION_LEVEL=1
    -DTEST_PSA_API=INITIAL_ATTESTATION

ninja -C . install -j 8

cd <trusted-firmware-m folder>
mkdir build_ns && cd build_ns
cmake -S /../trusted-firmware-m  /../tf-m-tests/tests_psa_arch -B . -GNinja
    -DCONFIG_SPE_PATH= /../build_s/api_ns -DTFM_TOOLCHAIN_FILE= /../build_s/api_ns/
↪cmake/toolchain_ns_ARMCLANG.cmake

ninja -C . -j 8
```

Copyright (c) 2021, Arm Limited. All rights reserved. Copyright (c) 2019, STMicroelectronics. All rights reserved. SPDX-License-Identifier: BSD-3-Clause

B_U585I_IOT02A

Discovery kit for IoT node with STM32U5 series https://www.st.com/en/evaluation-tools/B-U585I-IOT02A.html

Configuration and Build

GNUARM/ARMCLANG/IARARM compilation is available for this target. and build the selected configuration as follow.

The build configuration for TF-M is provided to the build system using command line arguments. Required arguments are noted below.

The following instructions build multi-core TF-M with regression test suites in Isolation Level 1.

In common STM (platform\ext\target\stm\common\build_stm) There are scripts that help users to build the TF-M project on all STM platforms

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
git checkout <recommended tf-m-tests commit>

mkdir build_s && cd build_s
```

(continues on next page)

(continued from previous page)

```
cmake -S /../tf-m-tests/tests_reg/spe -B . -GNinja -DTFM_PLATFORM=stm/b_u585i_iot02a
    -DTFM_TOOLCHAIN_FILE= /../toolchain_ARMCLANG.cmake
    -DCONFIG_TFM_SOURCE_PATH= /../trusted-firmware-m
    -DTFM_PSA_API=ON -DTFM_ISOLATION_LEVEL=1
    -DTEST_S=ON -DTEST_NS=ON

ninja -C . install -j 8

cd <trusted-firmware-m folder>
mkdir build_ns && cd build_ns
cmake -S /../trusted-firmware-m  /../tf-m-tests/tests_reg -B . -GNinja
    -DCONFIG_SPE_PATH= /../build_s/api_ns -DTFM_TOOLCHAIN_FILE= /../build_s/api_ns/
↪cmake/toolchain_ns_ARMCLANG.cmake

ninja -C . -j 8
```

The following instructions build multi-core TF-M with PSA API test suite for the attestation service in Isolation Level 1 on Linux.

```
cd <TF-M base folder>
cd <trusted-firmware-m folder>

git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
git checkout <recommended tf-m-tests commit>

mkdir build_s && cd build_s

cmake -S /../tf-m-tests/tests_psa_arch/spe -B . -GNinja -DTFM_PLATFORM=stm/b_u585i_iot02a
    -DTFM_TOOLCHAIN_FILE= /../toolchain_ARMCLANG.cmake
    -DCONFIG_TFM_SOURCE_PATH= /../trusted-firmware-m
    -DTFM_PSA_API=ON -DTFM_ISOLATION_LEVEL=1
    -DTEST_PSA_API=INITIAL_ATTESTATION

ninja -C . install -j 8

cd <trusted-firmware-m folder>
mkdir build_ns && cd build_ns
cmake -S /../trusted-firmware-m  /../tf-m-tests/tests_psa_arch -B . -GNinja
    -DCONFIG_SPE_PATH= /../build_s/api_ns -DTFM_TOOLCHAIN_FILE= /../build_s/api_ns/
↪cmake/toolchain_ns_ARMCLANG.cmake

ninja -C . -j 8
```

Copyright (c) 2021, STMicroelectronics. All rights reserved. SPDX-License-Identifier: BSD-3-Clause

---

Copyright (c) 2021, stmicroelectronics All rights reserved.

Copyright (c) 2020-2024, Arm Limited. All rights reserved.

Build instructions

Warning: The build process was changed a lot in Q3 2023 and included into the release v2.0. For building instructions for early versions please refer to the documentation of respective versions.

As you know from the introduction TF-M implements *SPE* with a set of secure services. TF-M application as *NSPE* client uses those services through isolation boundary via *PSA-FF-M* API. Both SPE and NSPE are separate binaries and built independently. SPE and NSPE binaries are combined and signed making the final image for downloading onto targets when building NSPE.

Note: This document describes the process of building a single SPE alone. Refer to *Building Tests* on how to build TF-M regression tests and PSA Arch tests to verify TF-M.

TF-M uses CMake **v3.15** or higher. Before starting please make sure you have all required software installed and configured as explained in the TF-M getting started.

Contents

- *Building TF-M (SPE)*
  - *Getting the source code*
  - *Configuring*
  - *Building binaries*
  - *Dependency management*
- *Building Application (NSPE)*
  - *SPE artifacts structure*
  - *NSPE toolchains*
  - *Basic SPE integration*

The additional building materials you can find in the following links:TF-M source folder

Build Tests

The tests is a TF-M application which verifies TF-M functionality on both SPE and NSPE sides. Thus, tests require an extension of SPE side with test code and extra functionality for some Non-Secure test cases. To inject that test code into SPE the CONFIG_TFM_TEST_DIR option is used. When SPE build system sees this option it adds the corresponding folder via add_subdirectory(${CONFIG_TFM_TEST_DIR} tf-m-tests) and includes it to SPE binary. Also, test configurations should be passed to SPE build to include building Secure Tests.

To hide these complexities to developers, TF-M implements a wrapper CMake in **tf-m-tests** repository[1] to build the SPE for testing rather than building it from the TF-M repository.

The recommended tf-m-tests repo commit to verify TF-M can be found at <TF-M source dir>/lib/ext/tf-m-tests/version.txt. It does not support auto-downloading as builds start from it. You need to download it manually before building any tests with the following commands:

```
git clone https://git.trustedfirmware.org/TF-M/tf-m-tests.git
git checkout <recommended tf-m-tests commit>
```

Regression Tests

For instructions on configuring, building and executing the regression tests please refer to the documentation in **tf-m-tests** repository (to be added). The regression test application is located under **/tests_reg** folder. It is recommended to build both SPE and NSPE from that folder.

The basic commands for building the regression tests will be:

[1] tf-m-tests

```
cd </tf-m-tests/tests_reg>
cmake -S spe -B build_spe -DTFM_PLATFORM=arm/mps2/an521 \
    -DCONFIG_TFM_SOURCE_PATH=<TF-M source dir> \
    -DTFM_TOOLCHAIN_FILE=<Absolute path to>/toolchain_ARMCLANG.cmake \
    -DTEST_S=ON -DTEST_NS=ON
cmake --build build_spe -- install

cmake -S . -B build_test -DCONFIG_SPE_PATH=<Absolute path to>/build_spe/api_ns
cmake --build build_test
```

Instead of enable all the supported Secure (TEST_S) and NS (TEST_NS) tests, you can also enable individual test suites by using -DTEST_S_<SUITE>=ON or -DTEST_NS_<SUITE>=ON. For the available test suites, refer to the default_s_test_config.cmake and default_ns_test_config.cmake files in tf-m-tests repo.

Note: All the test suite config options should be passed to the SPE build command, including NS ones. The SPE building command also accepts all the other config options used to build a single TF-M. All options passing to SPE build does not have to be duplicated in NSPE build, including NS test suite config options. And this also applies to the below *PSA API tests*.

PSA API tests

PSA API tests from https://github.com/ARM-software/psa-arch-tests use the same mechanism for SPE extension as the regression tests above utilising CONFIG_TFM_TEST_DIR option. PSA API tests are selected by the TEST_PSA_API variable. Enabling both regression tests and PSA API tests simultaneously is **not** supported.

TF-M implements a wrapper CMake for PSA API tests as well. The PSA API test codes are located under **/tests_psa_arch** folder.

Here is a brief description of the basic flow: Select one of the following test suites to be run.

```
-DTEST_PSA_API=INTERNAL_TRUSTED_STORAGE
-DTEST_PSA_API=PROTECTED_STORAGE
-DTEST_PSA_API=STORAGE
-DTEST_PSA_API=CRYPTO
-DTEST_PSA_API=INITIAL_ATTESTATION
-DTEST_PSA_API=IPC
```

Respectively for the corresponding service. For detailed information, please refer to **PSA Certified APIs Architecture Test Suite** section[2] in the documentation of **psa-arch-tests** repository.

For example, to enable the PSA API tests for the Crypto service:

```
cd </tf-m-tests/tests_psa_arch folder>
cmake -S spe -B build_spe -DTFM_PLATFORM=arm/mps2/an521 -DCONFIG_TFM_SOURCE_PATH=
↪<TF-M source dir>
    -DTEST_PSA_API=CRYPTO
cmake --build build_spe -- install

cmake -S . -B build_test -DCONFIG_SPE_PATH=<Absolute path to>/build_spe/api_ns
cmake --build build_test
```

Reference

Copyright (c) 2017-2024, Arm Limited. All rights reserved. Copyright (c) 2022, Cypress Semiconductor Corporation. All rights reserved.

[2] PSA Certified APIs Architecture Test Suite

Run TF-M tests and applications on Arm platforms

Instructions for how to run TF-M tests and applications on Arm platforms.

Follow *build instruction* to build the binaries. Follow *secure boot* to build the binaries with or without BL2 bootloader.

Execute TF-M regression tests on MPS2 boards and FVP

The BL2 bootloader and TF-M example application and tests have been verified using the reference model for MPS2 (AN521), in Keil MDK , Fixed Virtual Platforms and Arm Development Studio .

Note: The name of the reference model's executable can vary depending on toolchain.

- SMM-SSE-200 for Keil MDK
- FVP_MPS2_AEMv8M for Fixed Virtual Platforms and Arm Development Studio

For more information please refer to the appropriate toolchain's documentation: Keil MDK Documentation , Fixed Virtual Platforms Documentation , Arm Development Studio Documentation

---

To run the example code on an SSE-200 Fast-Model

Using FVP_MPS2_AEMv8M provided by Arm Development Studio 2019.1.

Example application and regression tests without BL2 bootloader

Add tfm_s.axf and tfm_ns.axf to symbol files in Debug Configuration menu.

```
<DS_PATH>/sw/models/bin/FVP_MPS2_AEMv8M \
--parameter fvp_mps2.platform_type=2 \
--parameter cpu0.baseline=0 \
--parameter cpu0.INITVTOR_S=0x10000000 \
--parameter cpu0.semihosting-enable=0 \
--parameter fvp_mps2.DISABLE_GATING=0 \
--parameter fvp_mps2.telnetterminal0.start_telnet=1 \
--parameter fvp_mps2.telnetterminal1.start_telnet=0 \
--parameter fvp_mps2.telnetterminal2.start_telnet=0 \
--parameter fvp_mps2.telnetterminal0.quiet=0 \
--parameter fvp_mps2.telnetterminal1.quiet=1 \
--parameter fvp_mps2.telnetterminal2.quiet=1 \
--application cpu0=<build_dir>/bin/tfm_s.axf \
--data cpu0=<build_dir>/bin/tfm_ns.bin@0x00100000
```

Example application and regression tests with BL2 bootloader

To test TF-M with bootloader, one must apply the following changes:

- Add bl2.axf to symbol files in DS-5 in Debug Configuration menu.
- Replace the last two lines of the previous command with this:

```
--application cpu0=<build_dir>/bin/bl2.axf \
--data cpu0=<build_dir>/tfm_s_ns_signed.bin@0x10080000
```

Test software upgrade with BL2 bootloader

BL2 bootloader is mandatory to test software update. Furthermore, two TF-M blobs must be built. Outputs of example application and regression test can be used to test it. Load output of example application to the primary slot (0x10080000) and output of regression test to the secondary slot (0x10180000). Add the following line to the end of the previous chapter:

```
--data cpu0=<build_dir>/tfm_s_ns_signed.bin@0x10180000
```

To run the example code on SSE 200 FPGA on MPS2 board

FPGA image is available to download here

To run BL2 bootloader and TF-M example application and tests in the MPS2 board, it is required to have SMM-SSE-200 for MPS2 (AN521) image in the MPS2 board SD card. The image should be located in <MPS2 device name>/MB/HBI0263<board revision letter>/AN521

The MPS2 board tested is HBI0263C referred also as MPS2+.

Warning: If you change the exe names, MPS2 expects file names in 8.3 format.

Example application

1. Copy bl2.bin and tfm_s_ns_signed.bin files from <build_dir>/bin to <MPS2 device name>/SOFTWARE/
2. Open <MPS2 device name>/MB/HBI0263C/AN521/images.txt
3. Update the AN521/images.txt file as follows:

```
TITLE: Versatile Express Images Configuration File
[IMAGES]
TOTALIMAGES: 2                ;Number of Images (Max: 32)
IMAGE0ADDRESS: 0x10000000
IMAGE0FILE: \Software\bl2.bin  ; BL2 bootloader
IMAGE1ADDRESS: 0x10080000
IMAGE1FILE: \Software\tfm_s_ns_signed.bin ; TF-M example application binary blob
```

4. Close <MPS2 device name>/MB/HBI0263C/AN521/images.txt
5. Unmount/eject the <MPS2 device name> unit
6. Reset the board to execute the TF-M example application
7. After completing the procedure you should be able to visualize on the serial port (baud 115200 8n1) the following messages:

```
[INF] Starting bootloader
[INF] Image 0: magic=good, copy_done=0xff, image_ok=0xff
[INF] Scratch: magic=bad, copy_done=0x5, image_ok=0xcf
[INF] Boot source: primary slot
[INF] Swap type: none
[INF] Bootloader chainload address offset: 0x80000
[INF] Jumping to the first image slot
[Sec Thread] Secure image initializing!
```

Regression tests

After completing the procedure you should be able to visualize on the serial port (baud 115200 8n1) the following messages:

```
[INF] Starting bootloader
[INF] Image 0: magic=good, copy_done=0xff, image_ok=0xff
[INF] Scratch: magic=bad, copy_done=0x5, image_ok=0xcf
[INF] Boot source: primary slot
[INF] Swap type: none
[INF] Bootloader chainload address offset: 0x80000
[INF] Jumping to the first image slot
```

(continues on next page)

(continued from previous page)

```
[Sec Thread] Secure image initializing!

#### Execute test suites for the protected storage service ####
Running Test Suite PS secure interface tests (TFM_PS_TEST_2XXX)...

> Executing 'TFM_PS_TEST_2001'
  Description: 'Create interface'
  TEST PASSED!
> Executing 'TFM_PS_TEST_2002'
  Description: 'Get handle interface (DEPRECATED)'
This test is DEPRECATED and the test execution was SKIPPED
  TEST PASSED!
> Executing 'TFM_PS_TEST_2003'
  Description: 'Get handle with null handle pointer (DEPRECATED)'
This test is DEPRECATED and the test execution was SKIPPED
  TEST PASSED!
> Executing 'TFM_PS_TEST_2004'
  Description: 'Write interface'
  TEST PASSED!
> Executing 'TFM_PS_TEST_2005'
  Description: 'Read interface'
....
```

Note: PS reliability tests take a few minutes to run on the MPS2.

Example application without BL2 bootloader

1. Copy tfm_s.bin and tfm_ns.bin files from <build_dir>/bin to <MPS2 device name>/SOFTWARE/
2. Open <MPS2 device name>/MB/HBI0263C/AN521/images.txt
3. Update the AN521/images.txt file as follows:

```
TITLE: Versatile Express Images Configuration File
[IMAGES]
TOTALIMAGES: 2                ;Number of Images (Max: 32)
IMAGE0ADDRESS: 0x10000000
IMAGE0FILE: \Software\tfm_s.bin  ; Secure code
IMAGE1ADDRESS: 0x00100000
IMAGE1FILE: \Software\tfm_ns.bin ; Non-secure code
```

4. Close <MPS2 device name>/MB/HBI0263C/AN521/images.txt
5. Unmount/eject the <MPS2 device name> unit
6. Reset the board to execute the TF-M example application
7. After completing the procedure you should be able to visualize on the serial port (baud 115200 8n1) the following messages:

```
[Sec Thread] Secure image initializing!
```

Regression tests without BL2 bootloader

After completing the procedure you should be able to visualize on the serial port (baud 115200 8n1) the following messages:

```
[Sec Thread] Secure image initializing!

#### Execute test suites for the protected storage service ####
Running Test Suite PS secure interface tests (TFM_PS_TEST_2XXX)...

> Executing 'TFM_PS_TEST_2001'
  Description: 'Create interface'
  TEST PASSED!
> Executing 'TFM_PS_TEST_2002'
  Description: 'Get handle interface (DEPRECATED)'
This test is DEPRECATED and the test execution was SKIPPED
  TEST PASSED!
> Executing 'TFM_PS_TEST_2003'
  Description: 'Get handle with null handle pointer (DEPRECATED)'
This test is DEPRECATED and the test execution was SKIPPED
  TEST PASSED!
> Executing 'TFM_PS_TEST_2004'
  Description: 'Write interface'
  TEST PASSED!
> Executing 'TFM_PS_TEST_2005'
  Description: 'Read interface'
....
```

Execute TF-M regression tests on Musca test chip boards

Note: Before executing any images on Musca-B1 board, please check the *target platform readme* to have the correct setup.

Install SRecord for Musca test chip boards:

- Windows: SRecord v1.63
- Linux: sudo apt-get install -y srecord

Example application with BL2 bootloader

1. Create a unified hex file comprising of both bl2.bin and tfm_s_ns_signed.bin for Musca-B1 or Musca-S1 platforms:

   ```
   srec_cat bin/bl2.bin -Binary -offset 0xA000000 bin/tfm_s_ns_signed.bin -Binary -offset 0xA020000 -o␣
   ↪tfm.hex -Intel
   ```

2. Power up the Musca board by connecting it to a computer with a USB lead. Press the PBON button if the green ON LED does not immediately turn on. The board should appear as a USB drive.
3. Copy tfm.hex to the USB drive. The orange PWR LED should start blinking.
4. Once the PWR LED stops blinking, power cycle or reset the board to boot from the new image.
5. After completing the procedure you should see the following messages on the DAPLink UART (baud 115200 8n1):

   ```
   [INF] Starting bootloader
   [INF] Image 0: magic=good, copy_done=0xff, image_ok=0xff
   [INF] Scratch: magic=bad, copy_done=0x5, image_ok=0xd9
   [INF] Boot source: primary slot
   [INF] Swap type: none
   [INF] Bootloader chainload address offset: 0x20000
   ```

(continues on next page)

(continued from previous page)

```
[INF] Jumping to the first image slot
[Sec Thread] Secure image initializing!
```

Regression tests with BL2 bootloader

Note: As the Internal Trusted Storage and Protected Storage tests use persistent storage, it is recommended to erase the storage area before running the tests. Existing data may prevent the tests from running to completion if, for example, there is not enough free space for the test data or the UIDs used by the tests have already been created with the write-once flag set. Repeated test runs can be done without erasing between runs.

To erase the storage when flashing an image, -fill 0xFF <start_addr> <end_addr> can be added to the srec_cat command used to create the combined hex file. The <start_addr> and <end_addr> are the start and end addresses of the storage area, found in the board's flash_layout.h file. The board's flash can also be erased via a debugger; see your IDE's documentation for instructions.

After completing the procedure you should see the following messages on the DAPLink UART (baud 115200 8n1):

```
[INF] Starting bootloader
[INF] Image 0: magic=good, copy_done=0xff, image_ok=0xff
[INF] Scratch: magic=bad, copy_done=0x5, image_ok=0x9
[INF] Boot source: primary slot
[INF] Swap type: none
[INF] Bootloader chainload address offset: 0x20000
[INF] Jumping to the first image slot
[Sec Thread] Secure image initializing!

#### Execute test suites for the protected storage service ####
Running Test Suite PS secure interface tests (TFM_PS_TEST_2XXX)...
> Executing 'TFM_PS_TEST_2001'
  Description: 'Create interface'
  TEST PASSED!
> Executing 'TFM_PS_TEST_2002'
  Description: 'Get handle interface (DEPRECATED)'
This test is DEPRECATED and the test execution was SKIPPED
  TEST PASSED!
> Executing 'TFM_PS_TEST_2003'
  Description: 'Get handle with null handle pointer (DEPRECATED)'
This test is DEPRECATED and the test execution was SKIPPED
  TEST PASSED!
> Executing 'TFM_PS_TEST_2004'
  Description: 'Get attributes interface'
  TEST PASSED!
> Executing 'TFM_PS_TEST_2005'
  Description: 'Get attributes with null attributes struct pointer'
....
```

PSA API tests

Follow the build instructions for the PSA API tests and then follow the above procedures for flashing the image to the board. The PSA API tests are linked into the TF-M binaries and will automatically run. A log similar to the following should be visible on the UART; it is normal for some tests to be skipped but there should be no failed tests:

```
[Sec Thread] Secure image initializing!
Booting TFM v1.1
```

(continues on next page)

(continued from previous page)

```
Non-Secure system starting...

***** PSA Architecture Test Suite - Version 1.0 *****

Running.. Storage Suite
****************************************

TEST: 401 | DESCRIPTION: UID not found check
[Info] Executing tests from non-secure

[Info] Executing ITS tests
[Check 1] Call get API for UID 6 which is not set
[Check 2] Call get_info API for UID 6 which is not set
[Check 3] Call remove API for UID 6 which is not set
[Check 4] Call get API for UID 6 which is removed
[Check 5] Call get_info API for UID 6 which is removed
[Check 6] Call remove API for UID 6 which is removed
[Check 7] Call get API for different UID 5
[Check 8] Call get_info API for different UID 5
[Check 9] Call remove API for different UID 5

[Info] Executing PS tests
[Check 1] Call get API for UID 6 which is not set
[Check 2] Call get_info API for UID 6 which is not set
[Check 3] Call remove API for UID 6 which is not set
[Check 4] Call get API for UID 6 which is removed
[Check 5] Call get_info API for UID 6 which is removed
[Check 6] Call remove API for UID 6 which is removed
[Check 7] Call get API for different UID 5
[Check 8] Call get_info API for different UID 5
[Check 9] Call remove API for different UID 5

TEST RESULT: PASSED

****************************************

<further tests removed from log for brevity>

************ Storage Suite Report **********
TOTAL TESTS     : 17
TOTAL PASSED    : 11
TOTAL SIM ERROR : 0
TOTAL FAILED    : 0
TOTAL SKIPPED   : 6
****************************************
```

Note: The Internal Trusted Storage and Protected Storage flash areas must be wiped before running the Storage test suites.

Many IDEs for embedded development (such as Keil µVision) offer a function to erase a device's flash. Refer to your IDE's documentation for instructions.

Another way this can be achieved is by using srec_cat with the -fill parameter to fill the corresponding area in the binary with the erase value of the flash (0xFF).

Refer to the platform flash layout for appropriate addresses to erase on other platforms.

This step is not required on targets that emulate flash storage in RAM, as it will be erased each time the device is reset. Note, however, that a warm reset may not clear SRAM contents, so it may be necessary to power the device off and on again between test runs.

Example application or regression tests on Musca-B1 without BL2 bootloader

Follow the above procedures, but create a unified hex file out of tfm_s.bin and tfm_ns.bin:

- Windows:

```
srec_cat.exe bin\tfm_s.bin -Binary -offset 0xA000000 bin\tfm_ns.bin -Binary -offset 0xA080000 -o tfm.
↪hex -Intel
```

- Linux:

```
srec_cat bin/tfm_s.bin -Binary -offset 0xA000000 bin/tfm_ns.bin -Binary -offset 0xA080000 -o tfm.hex␣
↪-Intel
```

Execute TF-M example and regression tests on MPS3 boards

To run the example code on CoreLink SSE-200 Subsystem for MPS3 (AN524)

FPGA image is available to download here

To run BL2 bootloader and TF-M example application and tests in the MPS3 board, it is required to have SMM-SSE-200 for MPS3 (AN524) image in the MPS3 board SD card. The image should be located in <MPS3 device name>/MB/HBI<BoardNumberBoardrevision>/AN524

And the current boot memory for AN524 is QSPI flash, so you need to set the correct REMAP option in <MPS3 device name>/MB/HBI<BoardNumberBoardrevision>/AN524/an524_v1.txt

```
REMAP: QSPI             ;REMAP boot device BRAM/QSPI.  Must match REMAPVAL below.
REMAPVAL: 1             ;REMAP register value e.g. 0-BRAM. 1-QSPI
```

The MPS3 board tested is HBI0309B.

Note: If you change the exe names, MPS3 expects file names in 8.3 format.

Example application

1. Copy bl2.bin and tfm_s_ns_signed.bin files from build dir to <MPS3 device name>/SOFTWARE/
2. Open <MPS3 device name>/MB/HBI0309B/AN524/images.txt
3. Update the images.txt file as follows:

```
TITLE: Arm MPS3 FPGA prototyping board Images Configuration File

[IMAGES]
TOTALIMAGES: 2                  ;Number of Images (Max: 32)

IMAGE0UPDATE: AUTO              ;Image Update:NONE/AUTO/FORCE
IMAGE0ADDRESS: 0x00000000       ;Please select the required executable program
IMAGE0FILE: \SOFTWARE\bl2.bin
IMAGE1UPDATE: AUTO
IMAGE1ADDRESS: 0x00040000
IMAGE1FILE: \SOFTWARE\tfm_s_ns_signed.bin
```

4. Close <MPS3 device name>/MB/HBI0309B/AN524/images.txt
5. Unmount/eject the <MPS3 device name> unit
6. Reset the board to execute the TF-M example application
7. After completing the procedure you should be able to visualize on the serial port (baud 115200 8n1) the following messages:

```
[INF] Starting bootloader
[INF] Image 0: magic= good, copy_done=0xff, image_ok=0xff
[INF] Scratch: magic=unset, copy_done=0x43, image_ok=0xff
[INF] Boot source: slot 0
[INF] Swap type: none
[INF] Bootloader chainload address offset: 0x40000
[INF] Jumping to the first image slot
[Sec Thread] Secure image initializing!
```

Note: If -DPLATFORM_DEFAULT_PROVISIONING=OFF is set then the provisioning bundle has to be placed on the 0x001CB000 address.

Regression tests

After completing the procedure you should be able to visualize on the serial port (baud 115200 8n1) the following messages:

```
[INF] Starting bootloader
[INF] Image 0: magic= good, copy_done=0xff, image_ok=0xff
[INF] Scratch: magic=unset, copy_done=0x9, image_ok=0xff
[INF] Boot source: slot 0
[INF] Swap type: none
[INF] Bootloader chainload address offset: 0x40000
[INF] Jumping to the first image slot
[Sec Thread] Secure image initializing!

#### Execute test suites for the Secure area ####
Running Test Suite PSA protected storage S interface tests (TFM_PS_TEST_2XXX)...
> Executing 'TFM_PS_TEST_2001'
  Description: 'Set interface'
  TEST PASSED!
> Executing 'TFM_PS_TEST_2002'
  Description: 'Set interface with create flags'
  TEST PASSED!
> Executing 'TFM_PS_TEST_2003'
  Description: 'Set interface with NULL data pointer'
  TEST PASSED!
> Executing 'TFM_PS_TEST_2004'
  Description: 'Set interface with invalid data length'
  TEST PASSED!
....
```

Note: Some of the attestation tests take a few minutes to run on the MPS3.

Example application without BL2 bootloader

1. Copy tfm_s.bin and tfm_ns.bin files from build dir to <MPS3 device name>/SOFTWARE/
2. Open <MPS3 device name>/MB/HBI0309B/AN524/images.txt
3. Update the images.txt file as follows:

```
TITLE: Arm MPS3 FPGA prototyping board Images Configuration File

[IMAGES]
TOTALIMAGES: 2                  ;Number of Images (Max: 32)

IMAGE0UPDATE: AUTO              ;Image Update:NONE/AUTO/FORCE
```

(continues on next page)

(continued from previous page)

```
IMAGE0ADDRESS: 0x00000000          ;Please select the required executable program
IMAGE0FILE: \SOFTWARE\tfm_s.bin
IMAGE1UPDATE: AUTO
IMAGE1ADDRESS: 0x000C0000
IMAGE1FILE: \SOFTWARE\tfm_ns.bin
```

4. Close <MPS3 device name>/MB/HBI0309B/AN521/images.txt
5. Unmount/eject the <MPS3 device name> unit
6. Reset the board to execute the TF-M example application
7. After completing the procedure you should be able to visualize on the serial port (baud 115200 8n1) the following messages:

```
[Sec Thread] Secure image initializing!
```

Regression tests without BL2 bootloader

After completing the procedure you should be able to visualize on the serial port (baud 115200 8n1) the following messages:

```
[Sec Thread] Secure image initializing!

#### Execute test suites for the Secure area ####
Running Test Suite PSA protected storage S interface tests (TFM_PS_TEST_2XXX)...
> Executing 'TFM_PS_TEST_2001'
  Description: 'Set interface'
  TEST PASSED!
> Executing 'TFM_PS_TEST_2002'
  Description: 'Set interface with create flags'
  TEST PASSED!
> Executing 'TFM_PS_TEST_2003'
  Description: 'Set interface with NULL data pointer'
  TEST PASSED!
> Executing 'TFM_PS_TEST_2004'
  Description: 'Set interface with invalid data length'
  TEST PASSED!
....
```

Firmware upgrade and image validation with BL2 bootloader

High level operation of BL2 bootloader and instructions for testing firmware upgrade is described in *secure boot*.

Copyright (c) 2017-2024, Arm Limited. All rights reserved.

Documentation generation

Two documents are available for generation:

- Reference Manual (HTML, PDF)
- User Guide (HTML, PDF)

The documentation build is independent from building the binary artifacts.

Tools and building environment

These tools are used to generate TF-M documentation:

- Doxygen v1.8.0 or later
- Graphviz dot v2.38.0 or later
- PlantUML v1.2018.11 or later
- Java runtime environment v1.8 or later (for running PlantUML)
- Sphinx and other python modules, listed in docs/requirements.txt

Additionally, for PDFs format:

- LaTeX
- PdfLaTeX

There are two ways of building TF-M reference manual:

1. As a custom target of TF-M CMake build system
2. Directly, using the command line tools

To prepare your building environment execute the following steps:

Linux

```
sudo apt-get install -y doxygen graphviz default-jre
mkdir ~/plantuml
curl -L http://sourceforge.net/projects/plantuml/files/plantuml.jar/download --output ~/plantuml/plantuml.
↪jar
export PLANTUML_JAR_PATH=~/plantuml/plantuml.jar

# For PDF generation
sudo apt-get install -y doxygen-latex

# Install the required Python modules
pip3 install --upgrade pip
cd trusted-firmware-m
pip3 install -r docs/requirements.txt
```

Windows

Download and install the following tools:

- Doxygen 1.8.8
- Graphviz 2.38
- **The Java runtime is part of the Arm DS installation or can be** downloaded from here
- PlantUML
- MikTeX - for PDF generation only

Set the environment variables, assuming that:

- doxygen, dot, and MikTeX binaries are available on the PATH.
- Java JVM is used from Arm DS installation.

```
set PLANTUML_JAR_PATH=<plantuml_Path>\plantuml.jar
set PATH=$PATH;<ARM_DS_PATH>\sw\java\bin

# Install the required Python modules
pip3 install --upgrade pip
cd trusted-firmware-m
pip3 install -r docs\requirements.txt
```

Build TF-M Reference Manual

The Reference Manual will be generated in the build_docs/reference_manual.

Linux

```
cd <TF-M base folder>
cmake -S docs -B build_docs
cmake --build build_docs -- tfm_docs_refman_html tfm_docs_refman_pdf
```

Windows

```
cd <TF-M base folder>
cmake -S docs -B build_docs -G"Unix Makefiles"
cmake --build build_docs -- tfm_docs_refman_html tfm_docs_refman_pdf
```

Build TF-M User Guide

The User Manual will be available under the directory build_docs/user_guide.

Linux

```
cd <TF-M base folder>
cmake -S docs -B build_docs
cmake --build build_docs -- tfm_docs_userguide_html tfm_docs_userguide_pdf
```

Windows

```
cd <TF-M base folder>
cmake -S docs -B build_docs -G"Unix Makefiles"
cmake --build build_docs -- tfm_docs_userguide_html tfm_docs_userguide_pdf
```

Direct build using a command line tools

The direct build will build both user_guide and reference_manual.

Linux

```
# Build the documentation from build_docs directory
cd <TF-M base folder>
mkdir build_docs
cp docs/conf.py build_docs/conf.py
cd build_docs
sphinx-build ./ user_guide
```

Windows

```
# Command line tools is yet not available for Windows
```

Dependencies

```
@startuml
 skinparam state {
   BackgroundColor #92AEE0
   FontColor black
   FontSize 16
   AttributeFontColor black
   AttributeFontSize 16
   BackgroundColor<<pdf>> #A293E2
   BackgroundColor<<doc>> #90DED6
 }
 state u_guide as "User Guide" <<doc>>
```

(continues on next page)

(continued from previous page)

```
state refman as "Reference Manual" <<doc>>
state rtd_theme as "sphinx-rtd-theme" <<doc>>
state tabs as "sphinx-tabs" <<doc>>
state sphnix_puml as "sphinxcontrib-plantuml" <<doc>>
state sphnix_svg as "sphinxcontrib-svg2pdfconverter" <<doc>>
state JRE as "JRE" <<doc>> : Java Runtime Environment
state gwiz as "Graphwiz dot" <<doc>>
state Sphinx as "Sphinx" <<doc>>
state python as "Python v3" <<doc>>
state m2r as "m2r2" <<doc>>
state PlantUML as "PlantUML" <<doc>>
state LaTex as "LaTex" <<pdf>>
state PdfLaTex as "PdfLaTex" <<pdf>>
state Doxygen as "Doxygen" <<doc>>
state librsvg as "librsvg2-bin" <<doc>>

[*] --> u_guide
u_guide --> Sphinx
Sphinx --> m2r
Sphinx --> rtd_theme
Sphinx --> tabs
Sphinx --> sphnix_puml
Sphinx --> sphnix_svg
m2r --> python
rtd_theme --> python
tabs --> python
sphnix_puml --> python
sphnix_svg --> python
sphnix_svg --> librsvg
Sphinx --> PlantUML
PlantUML --> JRE
PlantUML --> gwiz
Sphinx --> LaTex
LaTex --> PdfLaTex

[*] --> refman
refman --> Doxygen
Doxygen --> PlantUML
Doxygen --> LaTex
state Legend {
  state x as "For PDF generation only" <<pdf>>
}

@enduml
```

Copyright (c) 2017-2023, Arm Limited. All rights reserved.

Additional build instructions for the IAR toolchain

Follow the instructions in *software requirements*, but replace the -DTFM_TOOLCHAIN_FILE setting with toolchain_IARARM.cmake.

Notes for building with IARARM

IAR Embedded Workbench for ARM (EWARM) versions 8.42 or later are required.

Currently the MUSCA_B1 and MUSCA_S1 targets are not supported with IARARM, due to lack of testing.

cmake needs to be version 3.14 or newer.

The V8M IAR CMSIS_5 RTX libraries in CMSIS_5 5.5.0 has a problem and has been updated in CMSIS_5 5.7.0. The updated libraries are part of the tf-m-tests repo and no special instructions are needed when the libraries from this repo are used.

For all configurations and build options some of the QCBOR tests fail due to the tests not handling double float NaN:s according to the Arm Runtime ABI. This should be sorted out in the future.

Example: building TF-M for AN521 platform using IAR:

```
cd <TF-M base folder>
cmake -S . -B cmake_build -DTFM_PLATFORM=arm/mps2/an521 -DTFM_TOOLCHAIN_
↪FILE=toolchain_IARARM.cmake
cmake --build cmake_build -- install
```

Alternately using traditional cmake syntax

```
cd <TF-M base folder>
mkdir cmake_build
cd cmake_build
cmake .. -DTFM_PLATFORM=arm/mps2/an521 -DTFM_TOOLCHAIN_FILE=../toolchain_IARARM.
↪cmake
make install
```

Regression Tests for the AN521 target platform

```
cd <TF-M base folder>
cmake -S . -B cmake_build -DTFM_PLATFORM=arm/mps2/an521 -DTFM_TOOLCHAIN_
↪FILE=toolchain_IARARM.cmake -DTEST_S=ON -DTEST_NS=ON
cmake --build cmake_build -- install
```

Alternately using traditional cmake syntax

```
cd <TF-M base folder>
mkdir cmake_build
cd cmake_build
cmake .. -DTFM_PLATFORM=arm/mps2/an521 -DTFM_TOOLCHAIN_FILE=../toolchain_IARARM.
↪cmake -DTEST_S=ON -DTEST_NS=ON
make install
```

Copyright (c) 2020-2021, Arm Limited. All rights reserved.

Building TF-M (SPE)

This build generates the SPE binary and artifacts, necessary for *Building Application (NSPE)*.

Getting the source code

```
cd <base folder>
git clone https://git.trustedfirmware.org/TF-M/trusted-firmware-m.git
```

In this documentation, the cloned trusted-firmware-m repository will be referenced as <TF-M source dir>. Additionally, TF-M depends on several external projects, handled by CMake automatically but you can alter that behaviour using *Dependency management*.

Note:

- For building with Armclang compiler version 6.10.0+, please follow the note in TF-M getting started.
- For building with the IAR toolchain, please see the notes in *IAR software requirements*

- Please use "/" instead of "\" for paths when running CMAKE commands under Windows Command Prompt.

Configuring

TF-M has many config options for configuring and fine-tuning. Please check the *Configuration* section for the details. The **base** (default) configuration contains only essential components such as SPM and HW platform support hence the only mandatory argument to TF-M build is a platform name, provided via CMake command-line option -DTFM_PLATFORM=<platform name>, it can be:

- A relative path under <TF-M source dir>/platform/ext/target, for example arm/mps2/an521.
- An absolute path of target platform, mainly used for out-of-tree platform build.
- A basename of the target platform folder, for example an521.

Essential Directories

There are 3 essential directories used by CMake for building TF-M:

- Source code directory <TF-M source dir>
- Build directory <Build Dir> - the location of all intermediate files required to produce a build target.
- Install directory <Artifact Dir> - the location of the build output files.

Note::
It's recommended to use absolute paths for all directories. Relative paths may not fully work.

Toolchains

TF-M supports 3 toolchains for cross-compiling and building the project binaries:

- GNU - **default**
- ArmClang
- IAR

Each toolchain has a configuration file for the compiler and linker. They are located at the root directory of TF-M. Use TFM_TOOLCHAIN_FILE option to provide the absolute path to the preferred toolchain file, or relative path to working directory. The default **toolchain_GNUARM.cmake** is selected by *config_base.cmake* file if the option is omitted.

Build type

By default, a MinSizeRel configuration is built. Alternate build types can be specified with the CMAKE_BUILD_TYPE variable. The possible types are:

- Debug
- RelWithDebInfo
- Release
- MinSizeRel - **default**

Debug symbols are added by default to all builds, but can be removed from *Release* and *MinSizeRel* builds by setting TFM_DEBUG_SYMBOLS to *OFF*.

RelWithDebInfo, *Release* and *MinSizeRel* all have different optimizations turned on and hence will produce smaller, faster code than *Debug*. *MinSizeRel* will produce the smallest code and hence is often a good idea on RAM or flash-constrained systems.

Output files

In a successful build, a set of files will be created in the <Artifact Dir>. By default, it is <Build Dir>\api_ns subfolder but you can redirect the output to any location using CMAKE_INSTALL_PREFIX option. It can be an absolute path or relative to your current directory. For the contents of the artifact directory please refer to *SPE artifacts structure*.

Other build parameters

The full list of default options is in config/config_base.cmake and explained in *Build configuration*. Several important options are listed below.

Parameter	Description	Default value
BL2	Build level 2 secure bootloader.	ON
PROJECT_CONFIG_HEADER_FILE	User defined header file for TF-M config	
TFM_ISOLATION_LEVEL	Set TF-M isolation level.	1
TFM_PROFILE	See *TF-M Profiles*.	

Project Config Header File

CMake variable PROJECT_CONFIG_HEADER_FILE can be set by a user the full path to a configuration header file, which is used to fine-tune component options. The detailed reference for the project config header file is in *The Header File Config System*.

Building binaries

The command below shows a general template for building TF-M as a typical CMake project:

```
cmake -S <TF-M source dir> -B <Build Dir> -DTFM_PLATFORM=<platform>
cmake --build <Build Dir> -- install
```

Note: It is recommended to clean up the build directory before re-build if the config header file is updated. CMake is unable to automatically recognize the dependency when the header file is defined as a macro.

Building default configuration for an521

```
cd <TF-M source dir>
cmake -S . -B build -DTFM_PLATFORM=arm/mps2/an521
cmake --build build -- install
```

The command above is intended to do:

- take TF-M sources in the current . folder
- build SPE in the build folder
- for **an521** platform
- using GNU toolchain *by default*. Use -DTFM_TOOLCHAIN_FILE=<toolchain file> for alternatives as described in *Toolchains*
- install output files in build/api_ns folder *by default*. You can specify a different directory using -DCMAKE_INSTALL_PREFIX=<Artifact dir> as described in *Output files*

Note: It is recommended to build each different build configuration in a separate build directory.

CMake can generate code for many native build systems. TF-M is tested with Unix Makefiles (default) and Ninja. The -G option can specify alternative generators. For example for building with Ninja in the Debug *Build type* using ArmClang *Toolchains* you can use the following:

```
cd <TF-M source dir>
cmake -S . -B build -DTFM_PLATFORM=arm/mps2/an521 -GNinja -DTFM_TOOLCHAIN_
↪FILE=toolchain_ARMCLANG.cmake -DCMAKE_BUILD_TYPE=Debug
cmake --build build -- install
```

Dependency management

The TF-M build system will fetch all dependencies by default with appropriate versions and store them inside the build tree. In this case, the build tree location is <build_dir>/lib/ext.

If you have local copies already and wish to avoid having the libraries downloaded every time the build directory is deleted, then the following variables can be set to the paths to the root directories of the local repos. This will disable the automatic downloading for that dependencies and speed up development iterations or allow usage of a dependency version different from the current one. Additionally, these path variables can be set in localrepos.cmake file which will be included in a build if it exists. This file is ignored in TF-M git settings.

The following table lists the commonly used repos. For others, you can refer to lib/ext.

Dependency	Cmake variable	Git repo URL
Mbed Crypto	MBEDCRYPTO_PATH	https://github.com/ARMmbed/mbedtls
MCUboot	MCUBOOT_PATH	https://github.com/mcu-tools/mcuboot
QCBOR	QCBOR_PATH	https://github.com/laurencelundblade/QCBOR.git

The recommended versions of the dependencies are listed in config/config_base.cmake.

Note:

- Some repositories might need patches to allow building it as a part of TF-M. While these patches are being upstreamed they are stored in a dependency folder under lib/ext/. In order to use local repositories those patches shall be applied to original source. An alternative is to copy out the auto-downloaded repos under the <build_dir>/lib/ext. They have been applied with patches and can be used directly.

Example: building TF-M with local Mbed Crypto repo

Preparing a local repository consists of 2 steps: cloning and patching. This is only required to be done once. For dependencies without .patch files in their lib/ext directory the only required step is cloning the repo and checking out the correct branch.

```
cd <Mbed Crypto base folder>
git clone https://github.com/ARMmbed/mbedtls
cd mbedtls
git checkout <MBEDCRYPTO_VERSION> from <TF-M source dir>/config/config_base.cmake>
git apply <TF-M source dir>/lib/ext/mbedcrypto/*.patch
```

Note: <Mbed Crypto base folder> does not need to have any fixed position related to the TF-M repo so alternative method to get prepared dependency repos is to let TF-M download it once and then copy them out of the build/lib/ext folder.

Now build TF-M binaries

```
cd <TF-M source dir>
cmake -S . -B build -DTFM_PLATFORM=arm/mps2/an521 -DMBEDCRYPTO_PATH=<Mbed Crypto␣
↪base folder>/mbedtls
cmake --build build -- install
```

Building Application (NSPE)

As a result of *Building TF-M (SPE)* you will get a set of *Output files* in <Artifact Dir> required for building TF-M application. Essentially, SPE exports a binary and a set of C source files for PSA interface and platform. Please note that NSPE and SPE are independent projects and can be built using different toolchains and toolchain options.

SPE artifacts structure

SPE components prepared and installed for NSPE usage in <Artifact Dir> will have the following structure:

```
<Artifact Dir>
  bin
  cmake
  config
  image_signing
  interface
  platform
  CMakeLists.txt
```

With certain configurations, additional folders may also be installed. These folders have the following content:

- **bin** - binary images of SPE, Bootloader(optional) and combined.
- **cmake** - CMake scripts like SPE configuration and *NSPE toolchains*.
- **config** - Configuration files
- **image_signing** - binary image signing tool and keys.
- **interface** - PSA interface exposed by SPE.
- **platform** - source code for a selected hardware platform.
- **CMakeLists.txt** - CMake script for the artifacts integration in NSPE.

The content of <Artifact Dir> is an exported directory for integration with CMake projects.

Note: Attempting to change any file in <Artifact Dir> may cause incompatibility issues. Instead, please change the corresponding file in the <TF-M source dir>.

NSPE toolchains

SPE prepares and exports CMake toolchain files for building NSPE in all supported *Toolchains* in <Artifact Dir>/cmake folder. Toolchain used to build NSPE can be different from what is used to build SPE.

Basic SPE integration

Refer to the example of TF-M applications in **tf-m-extras** repository.

Copyright (c) 2017-2024, Arm Limited. All rights reserved. Copyright (c) 2022, Cypress Semiconductor Corporation. All rights reserved.

Configuration

Build configuration

All configuration options are provided by cmake variables, and their default values, with doc-strings, can be found in config/config_base.cmake.

Configuration is provided in multiple stages. Each stage will not override any config that has already been set at any of the prior stages.

1. Command-line variable settings are applied.
2. If the TFM_EXTRA_CONFIG_PATH variable has been set, that file is loaded.
3. If TEST_PSA_TEST is set, then PSA API test related config is applied from config/tests/config_test_psa_api.cmake.
4. If it exists, CMAKE_BUILD_TYPE specific config is applied from config/build_type/<build_type>.cmake.
5. Target specific config from platform/ext/target/<target_platform>/config.cmake is applied.
6. If CRYPTO_HW_ACCELERATOR is set, then a config specific to the accelerator type is applied if it exists.
7. If it exists, TFM Profile specific config is applied from config/profile/<tfm_profile>.cmake.
8. config/config_default.cmake is loaded.
9. If TEST_S or TEST_NS or other single test suite config like TEST_NS_ATTESTATION (see *Tests configuration*) is set, then config from ${TFM_TEST_REPO_PATH}/test/config/set_config.cmake and ${TFM_TEST_REPO_PATH}/test/config/default_ns_test_config.cmake or ${TFM_TEST_REPO_PATH}/test/config/default_s_test_config.cmake or ${TFM_TEST_REPO_PATH}/test/config/default_test_config.cmake is applied.

Warning: This means that command-line settings are not applied when they conflict with required platform settings. If it is required to override platform settings (this is not usually a good idea) then TFM_EXTRA_CONFIG_PATH should be used.

The Header File Config System

The header file configurations system is used to fine-tune component options.

The following diagram shows how the system works.

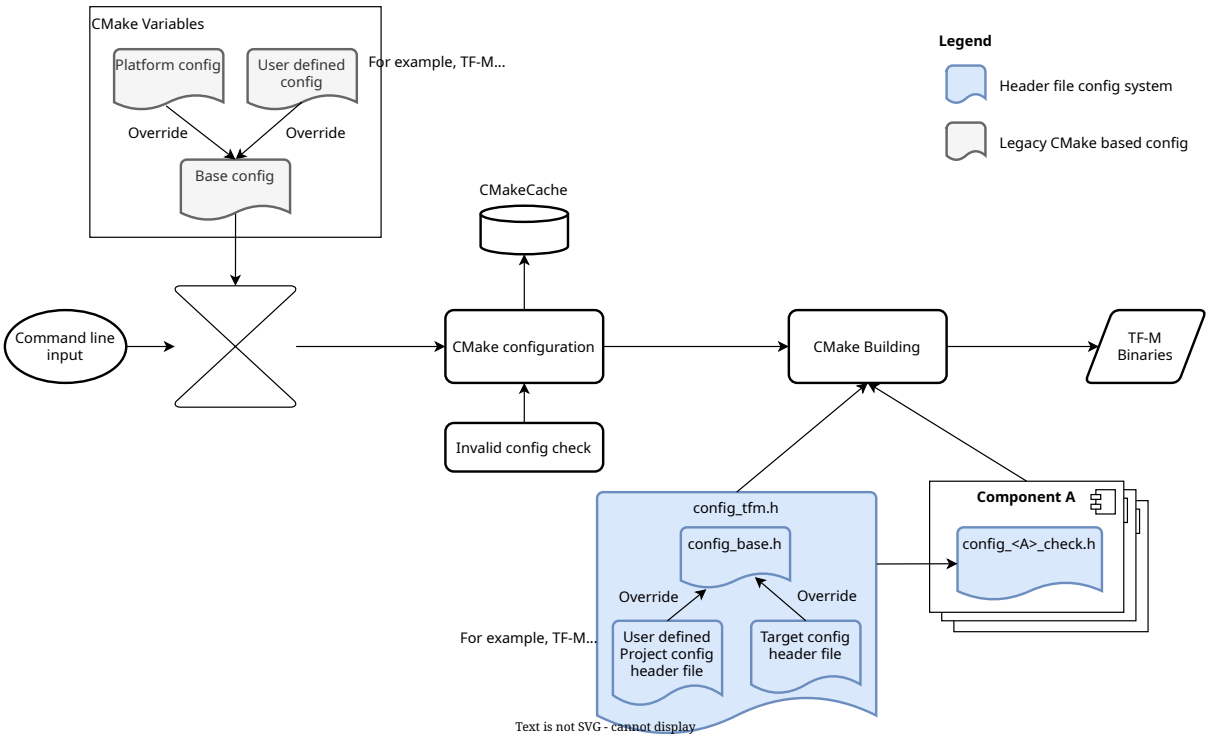
Source files shall include config_tfm.h when necessary to fetch Component option settings. It is expected that all Component options are included in config_tfm.h to explicitly set values for each option.

The config_tfm.h includes base configuration config_base.h. Refer to *Base Configuration* for details of the base configurations.

The config_tfm.h includes a customized project config file provided via compile definition PROJECT_CONFIG_HEADER_FILE. Customized Component options in the project config file overrides those configured in config_base.h. The project config header file can be

- Generated by the TF-M Kconfig system <kconfig_system>
- One of the header files of Profiles <tf-m_profiles>, set via the TFM_PROFILE build option.
- Manually customized profile based on pre-set profiles.

Users set CMake variable PROJECT_CONFIG_HEADER_FILE with the full path of the configuration header file.



A platform can adjust or place restriction on config options by providing a config_tfm_target.h under the root folder of their platforms. If the build system finds the file, it sets the TARGET_CONFIG_HEADER_FILE compile definition. Platform specific option settings in TARGET_CONFIG_HEADER_FILE overrides those configured in config_base.h.

```
#ifdef TARGET_CONFIG_HEADER_FILE
#include TARGET_CONFIG_HEADER_FILE
#endif

#ifdef PROJECT_CONFIG_HEADER_FILE
#include PROJECT_CONFIG_HEADER_FILE
#endif

#include "config_base.h"
```

Each component can implement a config_<comp_name>_check.h to validate component specific config dependencies. config_<comp_name>_check.h shall include config_tfm.h

---

Copyright (c) 2022, Arm Limited. All rights reserved.

The Kconfig System

The Kconfig system is an alternative tool to the CMake config system for users to change config options of TF-M.

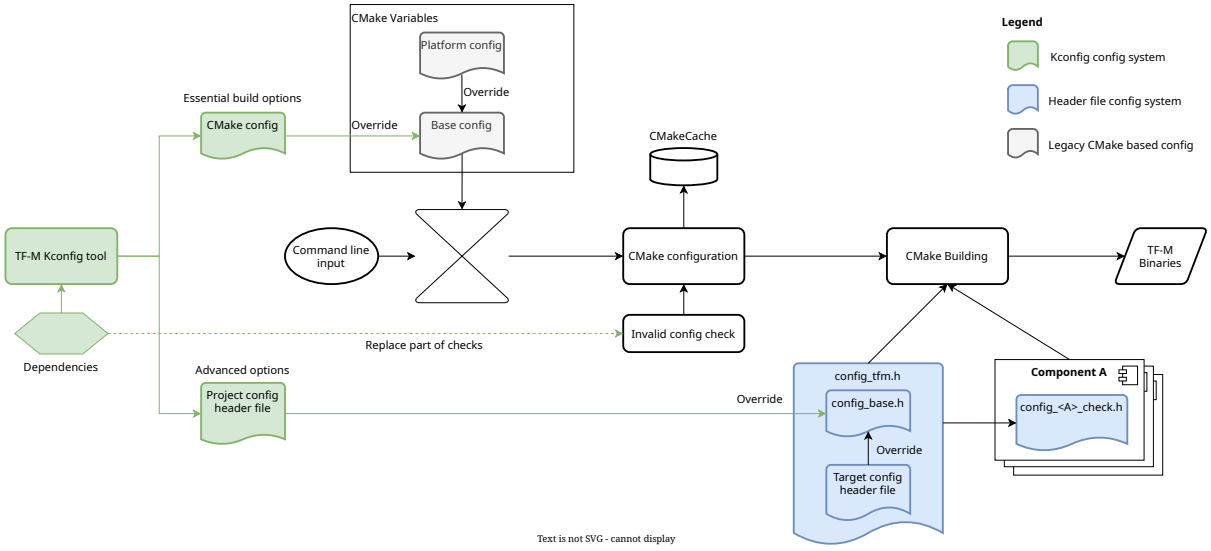
It handles dependencies and validations automatically when you change configurations so that the generated configuration options are always valid.

To use the Kconfig system, enable USE_KCONFIG_TOOL in command line. And enable MENUCONFIG to launch configuration GUI.

The Kconfig system consists of *The Kconfig tool* and the *The Kconfig files*.

The Kconfig tool

The Kconfig tool is a python script based on Kconfiglib to generate the following config files:



- CMake config file

  Contains CMake cache variables of building options.

- Header file

  Contains component options in the header file system. Component options are gathered together in a separate menu TF-M component configs in *The Kconfig files*.

- The .config and .config.old files

  The .config file which contains all the above configurations in the Kconfig format. It will be created after the first execution of the script. It is only used to allow users to make adjustments basing on the previous settings. The Kconfig tool will load it if it exists and .config.old will be created to save the previous configurations.

The tool supports loading multiple pre-set configuration files merging into a single one. The first loaded options are overridden by later ones if the config files contain duplicated options. And dependencies between config options are taken care of. It then launches a configuration GUI for users to change any config options if the MENUCONFIG is enabled in build command line.

Integration with TF-M build system

TF-M build system includes kconfig.cmake to integrate this tool. It prepares the parameters for the script and invokes it to load multiple configuration files basing on your build setup, including but not limited to

- Build type bound configurations, decided by CMAKE_BUILD_TYPE
- Profile configurations, decided by TFM_PROFILE

Customizing config options

By default, the Kconfig system only merges configuration files and generated the final config files. To customize the config options, there are several approaches.

Menuconfig

Menuconfig is the recommended approach to adjust the values of the config options because it has a graphic interface for you to easily change the options without worrying about dependencies.

To launch the menuconfig, you need to enable MENUCONFIG in addition to enabling USE_KCONFIG_TOOL.

```
cmake -S . -B cmake_build -DTFM_PLATFORM=arm/mps2/an521 \
                          -DUSE_KCONFIG_TOOL=ON \
                          -DMENUCONFIG=ON
```

Note: Although the Kconfiglib provides three menuconfig interfaces, only GUI menuconfig can be launched by CMake for the time being.

Command line options

The support of passing configurations via command line is kept for the Kconfig system.

```
cmake -S . -B cmake_build -DTFM_PLATFORM=arm/mps2/an521 \
                          -DUSE_KCONFIG_TOOL=ON \
                          -DTFM_ISOLATION_LEVEL=2
```

Kconfig file

You can also put the frequently used config options into a Kconfig file. When you need to apply the config options in that file, pass it via command line option -DKCONFIG_CONFIG_FILE

```
cmake -S . -B cmake_build -DTFM_PLATFORM=arm/mps2/an521 \
                          -DTFM_ISOLATION_LEVEL=2 \
                          -DUSE_KCONFIG_TOOL=ON \
                          -DKCONFIG_CONFIG_FILE=my_config.conf
```

Note: The command line set options override the ones in the config file. And you can always launch menuconfig to do the final adjustments.

The Kconfig files

The Kconfig files are the files written by the Kconfig language to describe config options. They also uses some Kconfiglib extensions such as optional source osource and relative source rsource so they can only work with the Kconfiglib.

Copyright (c) 2022-2023, Arm Limited. All rights reserved.

TF-M Profiles

The capabilities and resources may dramatically vary on different IoT devices. Some IoT devices may have very limited memory resource. The program on those devices should keep small memory footprint and basic functionalities. On the other hand, some devices may consist of more memory and extended storage, to support stronger software capabilities.

Diverse IoT use cases also require different levels of security and requirements on device resource. For example, use cases require different cipher capabilities. Selecting cipher suites can be sensitive to memory footprint on devices with constrained resource.

Trusted Firmware-M (TF-M) defines several general profiles, such as Profile Small, Profile Medium, Profile Medium ARoT-less and Profile Large, to provide different levels of security to fit diverse device capabilities and use cases applied on the top of the base configuration.

Each profile specifies a predefined list of features, targeting typical use cases with specific hardware constraints. Profiles can serve as reference designs, based on which developers can continue further development and configurations, according to use case.

TF-M Profiles align with Platform Security Architecture specifications and certification guidelines. It can help vendors to simplify security configuring for PSA certification.

Please check the table below to compare differences while details are discussed in the links below.

Trusted Firmware-M Profile Small Design

Introduction

As one of the TF-M Profiles, TF-M Profile Small (Profile S) consists of lightweight TF-M framework and basic Secure Services to keep smallest memory footprint, supporting fundamental security features on devices with ultra constrained resource.

This profile enables connecting with Edge Gateways and IoT Cloud Services supporting secure connection based solely on symmetric cryptography.

This document summarizes and discusses the features specified in TF-M Profile Small.

Overall design

TF-M Profile Small defines the following features:

- Lightweight framework
  - Secure Function (SFN) model[2]
  - Level 1 isolation
  - Buffer sharing allowed
  - Single secure context
- Crypto
  - Symmetric cipher only
  - Cipher suite for symmetric-key algorithms based protocols, such as cipher suites defined in TLS pre-shared key (TLS-PSK)[1].
    - Advanced Encryption Standard (AES) as symmetric crypto algorithm
    - SHA256 as Hash function
    - HMAC as Message Authentication Code algorithm
  - Only enable multi-part functions in hash, symmetric ciphers, Message Authentication Code (MAC) and Authenticated Encryption with Associated Data (AEAD) operations.
- Internal Trusted Storage (ITS)
  - No encryption
  - No rollback protection
  - Decrease internal transient buffer size
- Initial Attestation
  - Based on symmetric key algorithms
- Lightweight boot
  - Single image boot
  - Anti-rollback protection is enabled

[2] Arm Firmware Framework for M 1.1 Extensions
[1] Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)

Protected Storage, firmware update and other Secure Services provided by TF-M are disabled by default.

Design details

More details of TF-M Profile Small design are discussed in following sections.

Lightweight framework

TF-M framework model

SFN model is selected by default in Profile Small implementation. SFN model is defined in FF-M 1.1 extensions[Page 1209, 2]. It is a more simple implementation of TF-M framework and may also reduce memory footprint, compared with Inter-Process Communication (IPC) model[3].

Level 1 isolation

PSA Security Model[4] defines 3 levels of isolation.

- Level 1 isolation isolates Secure Processing Environment (SPE) from Non-secure Processing Environment (NSPE).
- PSA Root of Trust (PSA RoT) and Application Root of Trust (ARoT) are isolated from each other in level 2 isolation.
- Individual secure partitions are isolated from each other even within a particular security domain (PSA RoT, ARoT), in level 3 isolation.

Profile Small dedicated use cases with simple service model may not require level 2 or level 3 isolation. Devices which Profile Small aims at may be unable to implement stricter isolation, limited by hardware capabilities.

Level 1 isolation reduces requirements enforced by hardware isolation and cost of software for management.

Note: Security note

If a device or a use case enforces level 2 or level 3 isolation, it is suggested to apply other configurations, other than TF-M Profile Small.

Crypto service

TF-M Profile Small only requires symmetric crypto since symmetric algorithms require shorter keys and less computational burden, compared with asymmetric crypto.

By default, TF-M Profile Small requires the same capabilities as defined in TLS-PSK, to support symmetric key algorithms based protocols.

Note: Implementation note

Please note that TF-M Profile Small doesn't require that TLS-PSK is mandatory in applications. Instead, Profile Small only requires the same capabilities as defined in TLS-PSK, such as one symmetric cipher algorithm and one hash function.

TF-M Profile Small selects TLS-PSK cipher suite TLS_PSK_WITH_AES_128_CCM[5] as reference, which requires:

[3] Arm Platform Security Architecture Firmware Framework 1.0
[4] Platform Security Model 1.1
[5] AES-CCM Cipher Suites for Transport Layer Security (TLS)

- AES-128-CCM (AES CCM mode with 128-bit key) as symmetric crypto algorithm
- SHA256 as Hash function
- HMAC as Message Authentication Code algorithm

TLS_PSK_WITH_AES_128_CCM is selected since it requires small key length and less hardware capabilities, while keeping enough level of security.

Note: Implementation note

Developers can replace default algorithms with others or implement more algorithms.

Proper symmetric key algorithms and cipher suites should be selected according to device capabilities, the use case and the requirement of peers in connection.

Refer to *Crypto service configuration* for implementation details of configuring algorithms and cipher suites.

Note: Security note

It is recommended not to use MD5 or SHA-1 for message digests as they are subject to collision attacks[6][7].

By default, Profile Small only enables multi-part functions defined in PSA Cryptography API[13] in hash, symmetric ciphers, MAC and AEAD operations. Disabling single-part functions optimizes the code size of TF-M crypto service. Multi-part operations allows the message data to be processed in fragments instead of all at once. In static memory allocation, single-part operation may require to allocate a large memory space to support long message with unknown length. Therefore single-part operations can help users optimize memory footprint, especially while dealing with streaming data on IoT devices.

It may slightly increase the code size in applications to replace single-part implementation with mulit-part implementation. Althgouth the code size increment can be qaulified, if users are concerned about the code size increment, they can enable single-part operations by toggling Profile Small default configuration.

It may increase latency and overall time cost to implement cryptography functionality with single-part operations, compared to with multi-part ones. Users can enable single-part operations if the usage scenario requires single-part opreations to meet its perfermance metrics.

Secure Storage

TF-M Profile Small assumes that extremely constrained devices only contain basic on-chip storage, without external or removable storage. As a result, TF-M Profile Small includes ITS service and disables Protected Storage service.

Encryption and rollback protection

Neither encryption nor rollback protection is enabled in current ITS implementation.

It is expected that ITS relies solely on the physical inaccessibility property of on-chip storage, together with PSA isolation, without requiring additional cryptographic protection.

Internal transient buffer

[6] Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms
[7] Transitioning the Use of Cryptographic Algorithms and Key Lengths
[13] PSA Cryptography API 1.0

ITS implements a internal transient buffer[8] to hold the data read from/written to storage, especially for flash, to solve the alignment and security issues.

The internal transient buffer is aligned to the flash device's program unit. Copying data to it from the caller can align all write requests to the flash device's program unit. The internal transient buffer can help protect Flash access from some attacks, such as TOCTOU attack.

Although removing this internal buffer can save some memory consumption, typically 512 bytes, it may bring alignment or security issues. Therefore, to achieve a better trade-off between memory footprint and security, TF-M Profile Small optimizes the internal buffer size to 32 bytes by default.

As discussed in *Crypto service*, TF-M Profile Small requires AES-128 and SHA-256, which use 128-bit key and 256-bit key respectively. Besides, either long public/private keys or PKI-based certificates should be very rare as asymmetric crypto is not supported in Profile Small. Therefore, a 32-byte internal buffer should cover the assets in TF-M Profile Small use cases.

The buffer size can be adjusted according to use case and device Flash attributes. Refer to *Internal Trusted Storage configurations* for more details.

Initial Attestation

Profile Small requires an Initial Attestation secure service based on symmetric key algorithms. Refer to PSA Attestation API document[9] for details of Initial Attestation based on symmetric key algorithms.

It can heavily increase memory footprint to support Initial Attestation based on asymmetric key algorithms, due to asymmetric ciphers and related PKI modules.

Note: Implementation note

As pointed out by PSA Attestation API document[Page 1212, 9], the use cases of Initial Attestation based on symmetric key algorithms can be limited due to the associated infrastructure costs for key management and operational complexities. It may also restrict the ability to interoperate with scenarios that involve third parties.

If asymmetric key algorithms based Initial Attestation is required in use scenarios, it is recommended to select other TF-M Profiles which support asymmetric key algorithms.

Note: Implementation note

It is recommended to utilize the same MAC algorithm supported in Crypto service to complete the signing in COSE_Mac0, to minimize memory footprint.

Lightweight boot

If MCUBoot provided by TF-M is enabled, single image boot[10] is selected by default in Profile Small. In case of single image boot, secure and non-secure images are handled as a single blob and signed together during image generation.

However, secure and non-secure images must be updated together in single image boot. It may decrease the flexibility of image update and cost longer update process. Since the image sizes should usually be small with limited functionalities in Profile Small dedicated use case, the cost may still be reasonable.

[8] *ITS integration guide*
[9] PSA Attestation API 1.0 (ARM IHI 0085)
[10] *Secure boot*

BL2 implementation can be device specific. Devices may implement diverse boot processes with different features and configurations. However, anti-rollback protection is required as a mandatory feature of boot loader. Boot loader should be able to prevent unauthorized rollback, to protect devices from being downgraded to earlier versions with known vulnerabilities.

Implementation

Overview

The basic idea is to add dedicated profile CMake configuration files under folder config/profile for TF-M Profile Small default configuration.

The top-level Profile Small config file collects all the necessary configuration flags and set them to default values, to explicitly enable the features required in Profile Small and disable the unnecessary ones, during TF-M build.

A platform/use case can provide a configuration extension file to overwrite Profile Small default setting and append other configurations. This configuration extension file can be added via parameter TFM_EXTRA_CONFIG_PATH in build command line.

The behavior of the Profile Small build flow (particularly the order of configuration loading and overriding) can be found at *Build configuration*

The details of configurations will be covered in each module in *Implementation details*.

Implementation details

This section discusses the details of Profile Small implementation.

Top-level configuration files

The firmware framework configurations in config/profile/profile_small are shown below.

Table 28: TFM options in Profile Small top-level CMake config file

Configs	Default value	Descriptions
TFM_ISOLATION_LEVEL	1	Select level 2 isolation
TFM_PARTITION_INTER	ON	Enable ITS SP
ITS_BUF_SIZE	32	ITS internal transient buffer size
TFM_PARTITION_CRYPT	ON	Enable Crypto service
TFM_MBEDCRYPTO_CO	${CMAKE_SOURCE_DIR}/lib/ext/mbedcrypto/mbedcrypto_config/tfm_mbedcrypto_config_profile_small.h	Mbed Crypto config file path
TFM_MBEDCRYPTO_PS	${CMAKE_SOURCE_DIR}/lib/ext/mbedcrypto/mbedcrypto_config/crypto_config_profile_small.h	Mbed Crypto PSA config file path
CRYPTO_ASYM_SIGN_M	OFF	Enable asymmetric signature
CRYPTO_ASYM_ENCRYI	OFF	Enable asymmetric encryption
TFM_PARTITION_INITIA	ON	Enable Initial Attestation service
SYMMETRIC_INITIAL_ATTESTAT	ON	Enable symmetric attestation
TFM_PARTITION_PROTI	OFF	Enable PS service
TFM_PARTITION_PLATF	OFF	Enable TF-M Platform SP

Note: Implementation note

The following sections focus on the feature selection via configuration setting. Dedicated optimization on memory footprint is not covered in this document.

Device configuration extension

To change default configurations and add platform specific configurations, a platform can add a platform configuration file at platform/ext<TFM_PLATFORM>/config.cmake

TF-M framework setting

The top-level Profile Small CMake config file selects SFN model and level 1 isolation.

In SFN model, -DPSA_FRAMEWORK_HAS_MM_IOVEC is enabled by default. It reduces memory footprint by avoiding the transient copy from input vectors and copy to output vectors.

Crypto service configuration

Crypto Secure Partition

TF-M Profile Small enables Crypto Secure Partition (SP) in its top-level CMake config file. Crypto SP modules not supported in TF-M Profile Small are disabled. The disabled modules/features are shown below.

- Disable asymmetric cipher
- Disable single-part operations in Hash, MAC, AEAD and symmetric ciphers via selecting CRYPTO_SINGLE_PART_FUNCS_DISABLED

Other modules and configurations[11] are kept as default values.

Additional configuration flags with more fine granularity can be added to control building of specific crypto algorithms and corresponding test cases.

Mbed Crypto configurations

TF-M Profile Small adds a dedicated Mbed Crypto config file tfm_mbedcrypto_config_profile_small.h and Mbed Crypto PSA config file crypto_config_profile_small.h at /lib/ext/mbedcrypto/mbedcrypto_config folder, instead of the common one tfm_mbedcrypto_config_default.h and crypto_config_default.hPage 1215, 11.

Major Mbed Crypto configurations are set as listed below:

- Enable SHA256
- Enable generic message digest wrappers
- Enable AES
- Enable CCM mode for symmetric ciphers
- Disable other modes for symmetric ciphers
- Disable asymmetric ciphers
- Disable HMAC-based key derivation function (HKDF)

Other configurations can be selected to optimize the memory footprint of Crypto module.

A device/use case can append an extra config header to the Profile Small default Mbed Crypto config file. This can be done by setting the TFM_MBEDCRYPTO_PLATFORM_EXTRA_CONFIG_PATH cmake variable in the platform config file platform/ext<TFM_PLATFORM>/config.cmake. This cmake variable is a wrapper around the MBEDTLS_USER_CONFIG_FILE options, but is preferred as it keeps all configuration in cmake.

Internal Trusted Storage configurations

ITS service is enabled in top-level Profile Small CMake config file.

The internal transient buffer size ITS_BUF_SIZEPage 1212, 8 is set to 32 bytes by default. A platform/use case can overwrite the buffer size in its specific configuration extension according to its actual requirement of assets and Flash attributes.

Profile Small CMake config file won't touch the configurations of device specific Flash hardware attributesPage 1212, 8.

Initial Attestation secure service

TF-M Profile Small provides a reference implementation of symmetric key algorithms based Initial Attestation, using HMAC SHA-256 as MAC algorithm in COSE_Mac0 structure. The implementation follows PSA Attestation API documentPage 1212, 9.

Profile Small top-level config file enables Initial Attestation secure service and selects symmetric key algorithms based Initial Attestation by default.

[11] *Crypto design*

- Set TFM_PARTITION_INITIAL_ATTESTATION to ON
- Set SYMMETRIC_INITIAL_ATTESTATION to ON

Symmetric and asymmetric key algorithms based Initial Attestation can share the same generations of token claims, except Instance ID claim.

Profile Small may implement the procedure or rely on a 3rd-party tool to construct and sign COSE_Mac0 structure.

Details of symmetric key algorithms based Initial Attestation design will be covered in a dedicated document.

Disabled secure services

Protected Storage and Platform Service are disabled by default in Profile Small top-level CMake config file.

Test configuration

Some cryptography tests are disabled due to the reduced Mbed Crypto config. Some of them are shown in the table below.

Table 29: TFM options in Profile Small top-level CMake config file

Configs	Default value	Descriptions
TFM_CRYPTO_TEST_ALG_CB	OFF	Test CBC cryptography mode
TFM_CRYPTO_TEST_ALG_CC	ON	Test CCM cryptography mode
TFM_CRYPTO_TEST_ALG_CF	OFF	Test CFB cryptography mode
TFM_CRYPTO_TEST_ALG_EC	OFF	Test ECB cryptography mode
TFM_CRYPTO_TEST_ALG_CT	OFF	Test CTR cryptography mode
TFM_CRYPTO_TEST_ALG_OF	OFF	Test OFB cryptography mode
TFM_CRYPTO_TEST_ALG_GC	OFF	Test GCM cryptography mode
TFM_CRYPTO_TEST_ALG_SH	OFF	Test SHA-384 cryptography algorithm
TFM_CRYPTO_TEST_ALG_SH	OFF	Test SHA-512 cryptography algorithm
TFM_CRYPTO_TEST_HKDF	OFF	Test HKDF key derivation algorithm
TFM_CRYPTO_TEST_ECDH	OFF	Test ECDH key agreement algorithm
TFM_CRYPTO_TEST_CHACHA	OFF	Test ChaCha20 stream cipher
TFM_CRYPTO_TEST_CHACHA	OFF	Test ChaCha20-Poly1305 AEAD algorithm
TFM_CRYPTO_TEST_SINGLE_	OFF	Test single-part operations in hash, MAC, AEAD and symmetric ciphers

BL2 setting

Profile Small enables MCUBoot provided by TF-M by default. A platform can overwrite this configuration by disabling MCUBoot in its configuration extension file platform/ext<TFM_PLATFORM>/config.cmake.

If MCUBoot provided by TF-M is enabled, single image boot is selected in TF-M Profile Small top-level CMake config file.

If a device implements its own boot loader, the configurations are implementation defined.

Table 30: BL2 options in Profile Small top-level CMake config file

Configs	Default value	Descriptions
BL2	ON	Enable MCUBoot bootloader
MCUBOOT_IMAGE_NUMBER	1	Combine S and NS images

Platform support

Building Profile Small

To build Profile Small, argument TFM_PROFILE in build command line should be set to profile_small.

Take AN521 as an example.

The following commands build Profile Small without test cases on **AN521** with build type **MinSizeRel**, built by **Armclang**. SFN model is selected by default.

```
cd <TFM root dir>
mkdir build && cd build
cmake -DTFM_PLATFORM=arm/mps2/an521 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTFM_PROFILE=profile_small \
      -DCMAKE_BUILD_TYPE=MinSizeRel \
      ../
cmake --build ./ -- install
```

The following commands build Profile Small with regression test cases on **AN521** with build type **MinSizeRel**, built by **Armclang**. SFN model is selected by default.

```
cd <TFM root dir>
mkdir build && cd build
cmake -DTFM_PLATFORM=arm/mps2/an521 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTFM_PROFILE=profile_small \
      -DCMAKE_BUILD_TYPE=MinSizeRel \
      -DTEST_NS=ON \
      ../
cmake --build ./ -- install
```

Note:

- For devices with more constrained memory and flash requirements, it is possible to build with either only TEST_S enabled or only TEST_NS enabled. This will decrease the size of the test images. Note that both test suites must still be run to ensure correct operation.

More details of building instructions and parameters can be found TF-M build instruction guide[12].

Reference

Copyright (c) 2020-2022, Arm Limited. All rights reserved.

Trusted Firmware-M Profile Medium-ARoT-less

Introduction

TF-M Profile Medium-ARoT-less is a reference implementation to align with security requirements defined in PSA Certified ARoT-less Level 2 protection profile (PSA Certified ARoT-less)[1].

[12] *TF-M build instruction*

[1] SESIP Profile for PSA Certified ARoT-less Level 2

TF-M Profile Medium-ARoT-less is defined based on TF-M Profile Medium[2], which aligns with PSA Certified Level 2 Protection Profile[3].

Overall design

TF-M Profile Medium-ARoT-less defines the following feature set:

- Firmware Framework
  - Secure Function (SFN) model[4]
  - Isolation level 1[5]
- Internal Trusted Storage (ITS)
- Crypto
  - Support both symmetric cryptography and asymmetric cryptography
  - Asymmetric key based cipher suite suggested in TLS/DTLS profiles for IoT[6] and CoAP[7], including
    - Authenticated Encryption with Associated Data (AEAD) algorithm
    - Asymmetric key algorithm based signature and verification
    - Public-key cryptography based key exchange
    - Hash function
    - HMAC for default Pseudorandom Function (PRF)
  - Asymmetric digital signature and verification for Initial Attestation Token (IAT)
- Initial Attestation
  - Asymmetric key algorithm based Initial Attestation
- Lightweight boot
  - Anti-rollback protection
  - Multiple image boot
- Firmware Update (FWU) RoT Service

Design details

Most of design in Profile Medium-ARoT-less is identical to that in Profile Medium. Refer to Profile Medium document[Page 1218, 2] for details. Only the differences between Profile Medium-ARoT-less and Profile Medium are specified below.

Firmware framework

PSA Certified ARoT-less[Page 1217, 1] is only applicable to devices that don't support Application RoT (ARoT) services. The platform only consists of PSA RoT domain(s) in SPE making it unnecessary to implement the isolation between ARoT and PSA RoT. Therefore, this profile selects isolation level 1 to simplify implementation and optimize memory footprint and performance.

Since only isolation level 1 is required, this profile enables SFN model rather than IPC mode for further simplification.

[2] *Trusted Firmware-M Profile Medium Design*
[3] SESIP Profile for PSA Certified Level 2
[4] Arm Firmware Framework for M 1.1 Extensions
[5] Arm Platform Security Architecture Firmware Framework 1.0
[6] Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things
[7] The Constrained Application Protocol (CoAP)

Protected Storage

Protected Storage (PS) is implemented as an Application RoT service in TF-M by default. Therefore, PS is disabled by default in this profile.

FWU RoT Service

As PSA Certified ARoT-less requests, FWU RoT Service is enabled by default to support secure update of platform if the platform supports FWU.

Note: Implementation note

The entire secure update sequence involves multiple agents and components, including bootloader, TF-M FWU RoT Service, image update application(s), remote server(s), etc.

The secure update sequence is implementation-defined. Here is a reference of TF-M integration with FreeRTOS OTA[8].

Implementation

The basic idea is to add dedicated profile CMake configuration files config/profile/profile_medium_arotless for this profile default configuration.

This top-level config file collects all the fundamental configuration flags and set them to default values, to explicitly configured the features required in this profile, during TF-M build.

A platform/use case overwrite the default settings to configure this profile.

The behavior of the build flow (particularly the order of configuration loading and overriding) can be found at *Build configuration*.

The default configurations in config/profile/profile_medium_arotless are shown below.

[8] Secure OTA Updates for Cortex-M Devices with FreeRTOS

Table 31: Config flags in Profile Medium-ARoT-less top-level CMake config file

Configs	Default value	Descriptions
TFM_ISOLATION_LEV	1	Select level 1 isolation
CONFIG_TFM_SPM_BACK	SFN	Select SFN model
TFM_PARTITION_INTI	ON	Enable ITS SP
ITS_BUF_SIZE	32	ITS internal transient buffer size
TFM_PARTITION_CRY	ON	Enable Crypto service
CRYPTO_ASYM_ENCR	OFF	Enable Crypto asymmetric encryption operations
TFM_MBEDCRYPTO_C	${CMAKE_SOURCE_DIR}/lib/ext/mbedcrypto/mbedcrypto_config/tfm_mbedcrypto_config_profile_medium.h	Re-use Profile Medium configuration
TFM_MBEDCRYPTO_F	${CMAKE_SOURCE_DIR}/lib/ext/mbedcrypto/mbedcrypto_config/crypto_config_profile_medium.h	Re-use Profile Medium configuration
TFM_PARTITION_INIT	ON	Enable Initial Attestation service
TFM_PARTITION_FIRI	ON	Enable Firmware Update service
TFM_PARTITION_PRO	OFF	Disable PS service
TFM_PARTITION_PLA	ON	Enable TF-M Platform SP

Note: Where a configuration is the same as the default in config/config_default.cmake, it might be omitted from the profile configuration file.

Note: Implementation note

If the platform doesn't support secure update functionalities, FWU service will be disabled automatically. A warning will be thrown out during build.

Regression test configuration

FWU regression tests and SFN regression tests are selected by default when regression tests are enabled. Other implementations are the same as those in Profile Medium.

Platform support

To enable Profile Medium-ARoT-less on a platform, the platform specific CMake file should be added into the platform support list in top-level Profile Medium-ARoT-less CMake config file.

Building Profile Medium-ARoT-less

To build Profile Medium-ARoT-less, argument TFM_PROFILE in build command line should be set to profile_medium_arotless.

Take Musca-B1 as an example. The following commands build Profile Medium-ARoT-less without test cases on **Musca-B1** with build type **MinSizeRel**, built by **Armclang**.

```
cd <TFM root dir>
mkdir build && cd build
cmake -DTFM_PLATFORM=musca-b1 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTFM_PROFILE=profile_medium_arotless \
      -DCMAKE_BUILD_TYPE=MinSizeRel \
      ../
cmake --build ./ -- install
```

The following commands build Profile Medium-ARoT-less with regression test cases on **Musca-B1** with build type **MinSizeRel**, built by **Armclang**.

```
cd <TFM root dir>
mkdir build && cd build
cmake -DTFM_PLATFORM=musca-b1 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTFM_PROFILE=profile_medium_arotless \
      -DCMAKE_BUILD_TYPE=MinSizeRel \
      -DTEST_S=ON -DTEST_NS=ON \
      ../
cmake --build ./ -- install
```

Note:

- For devices with more constrained memory and flash requirements, it is possible to build with either only TEST_S enabled or only TEST_NS enabled. This will decrease the size of the test images. Note that both test suites must still be run to ensure correct operation.

More details of building instructions and parameters can be found TF-M build instruction guide[9].

Reference

Copyright (c) 2020-2023, Arm Limited. All rights reserved.

Trusted Firmware-M Profile Medium Design

Introduction

Compared with Profile Small, Profile Medium aims to securely connect devices to Cloud services with asymmetric cipher support. Profile Medium target devices need more resources for more cipher algorithms and higher isolation levels.

Overall design

TF-M Profile Medium defines the following feature set:

- Firmware Framework
  - Inter-Process Communication (IPC) model [PSA-FF-M]
  - Isolation level 2 [PSA-FF-M]
- Internal Trusted Storage (ITS)
- Crypto
  - Support both symmetric cryptography and asymmetric cryptography

[9] *TF-M build instruction*

  - Asymmetric key based cipher suite suggested in TLS/DTLS profiles for IoT [RFC7925] and CoAP [RFC7252], including
    * Authenticated Encryption with Associated Data (AEAD) algorithm
    * Asymmetric key algorithm based signature and verification
    * Public-key cryptography based key exchange
    * Hash function
    * HMAC for default Pseudorandom Function (PRF)
  - Asymmetric digital signature and verification for Initial Attestation Token (IAT)
- Initial Attestation
  - Asymmetric key algorithm based Initial Attestation
- Lightweight boot
  - Anti-rollback protection
  - Multiple image boot
- Protected Storage (PS) if off-chip storage device is integrated
  - Data confidentiality
  - Data integrity
  - Rollback protection

Design details

More details of TF-M Profile Medium design are described in following sections.

Firmware framework

Profile Medium with IPC model and isolation level 2 aims to support usage scenarios which require more complicated secure service model and additional protection to PSA RoT.

Level 2 isolation

Profile Medium selects isolation level 2 by default. In addition to isolation level 1, the PSA Root of Trust (PSA RoT) is also protected from access by the Application Root of Trust (App RoT) in level 2 isolation.

IPC model

Profile Medium enables IPC model by default. IPC model can achieve a more flexible framework and higher levels of isolation, but may require more memory footprint and bring in longer latency, compared to SFN model.

TF-M IPC model implementation follows the PSA Firmware Framework for M (PSA-FF-M) [PSA-FF-M].

Crypto service

Compared to Profile Small, Profile Medium includes asymmetric cryptography to support direct connection to Cloud services via common protocols, such as TLS/DTLS 1.2.

As suggested in CoAP [RFC7252] and [RFC7925], TF-M Profile Medium by default selects TLS_ECDHE_ECDSA_WITH_AES_128_CCM as reference, which requires:

- ECDHE_ECDSA as key exchange algorithm.

- AES-128-CCM (AES CCM mode with 128-bit key) as AEAD algorithm. Platforms can implement AES-128-CCM with truncated authentication tag to achieve less network bandwidth [RFC7925].
- SHA256 as Hash function.
- HMAC as Message Authentication Code algorithm.

Applications can also support TLS PSK [RFC4279] cipher suites, such as TLS_PSK_WITH_AES_128_CCM [RFC7925].

Note: Implementation note

Developers can replace default algorithms with others or implement more algorithms according to actual usage scenarios and device capabilities.

If a Crypto hardware accelerator is integrated, the cipher suites and algorithms also depend on those accelerator features.

More details of cipher suite are described below.

Digital signature and verification

ECDSA is selected by default in Profile Medium. ECDSA requires much shorter keys compared with RSA at the same security level. Therefore, ECDSA can cost less storage area for assets and less network bandwidth to setup a TLS connection. ECDSA is also preferred for forward compatibility of future TLS versions.

As requested in [RFC7251], ECC curve secp256r1 should be supported. More ECC curves can be added based on the requirements in production.

If usage scenarios require RSA algorithm for backward compatibility and legacy applications, platforms can add RSA support or replace ECDSA with RSA. The cipher suite should be switched accordingly.

AEAD algorithm

If Protected Storage (PS) is implemented, it is recommended to select the same AEAD algorithm for PS service as the one used by TLS/DTLS cipher suite.

Internal Trusted Storage

The configuration of ITS is the same as those in Profile Small [PROFILE-S].

Lightweight boot

BL2 implementation can be device specific. Devices may implement diverse boot processes with different features and configurations. However, the boot loader must support anti-rollback protection. Boot loader must be able to prevent unauthorized rollback, to protect devices from being downgraded to earlier versions with known vulnerabilities.

MCUBoot in TF-M is configured as multiple image boot by default in Profile Medium. In multiple image boot, secure and non-secure images can be signed independently with different keys and they can be updated separately. It can support multiple vendors scenarios, in which non-secure and secure images are generated and updated by different vendors. Multiple image boot may require more storage area compared with single image boot.

Protected Storage

PS service is required if an off-chip storage device is integrated and used on the platform.

TF-M PS service relies on an AEAD algorithm to ensure data confidentiality and integrity. It is recommended to select the same AEAD algorithm as the one used for TLS/DTLS cipher suite.

Anti-rollback protection in PS relies on non-volatile counter(s) provided by TF-M Platform Secure Partition (SP).

Implementation

Overview

The basic idea is to add dedicated profile CMake configuration files under folder config/profile for TF-M Profile Medium default configuration, the same as Profile Small does.

The top-level Profile Medium config file collects all the necessary configuration flags and set them to default values, to explicitly enable the features required in Profile Medium and disable the unnecessary ones, during TF-M build.

A platform/use case can provide a configuration extension file to overwrite Profile Medium default setting and append other configurations. This configuration extension file can be added via parameter TFM_EXTRA_CONFIG_PATH in build command line.

The behaviour of the Profile Medium build flow (particularly the order of configuration loading and overriding) can be found at *Build configuration*

The details of configurations will be covered in each module in *Implementation details*.

Implementation details

This section discusses the details of Profile Medium implementation.

Top-level configuration files

The firmware framework configurations in config/profile/profile_medium are shown below.

Table 32: Config flags in Profile Medium top-level CMake config file

Configs	Default value	Descriptions
TFM_ISOLATION_LEV	2	Select level 2 isolation
TFM_PARTITION_INTI	ON	Enable ITS SP
ITS_BUF_SIZE	32	ITS internal transient buffer size
TFM_PARTITION_CRY	ON	Enable Crypto service
CRYPTO_ASYM_ENCR	OFF	Enable Crypto asymmetric encryption operations
TFM_MBEDCRYPTO_C	${CMAKE_SOURCE_DIR}/lib/ext/mbedcrypto/mbedcrypto_config/tfm_mbedcrypto_config_profile_medium.h	Mbed Crypto config file path
TFM_MBEDCRYPTO_I	${CMAKE_SOURCE_DIR}/lib/ext/mbedcrypto/mbedcrypto_config/crypto_config_profile_medium.h	Mbed Crypto PSA config file path
TFM_PARTITION_INIT	ON	Enable Initial Attestation service
TFM_PARTITION_PRO	ON	Enable PS service
TFM_PARTITION_PLA	ON	Enable TF-M Platform SP

Note: Where a configuration is the same as the default in config/config_base.cmake, it is omitted from the profile configuration file.

Test configuration

Standard regression test configuration applies. This means that enabling regression testing via

-DTEST_S=ON -DTEST_NS=ON

Will enable testing for all enabled partitions. See above for details of enabled partitions. Because Profile Medium enables IPC model, the IPC tests are also enabled.

Some cryptography tests are disabled due to the reduced Mbed Crypto config.

Table 33: TFM options in Profile Medium top-level CMake config file

Configs	Default value	Descriptions
TFM_CRYPTO_TEST_ALG_CB	OFF	Disable CBC mode test
TFM_CRYPTO_TEST_ALG_CC	ON	Enable CCM mode test
TFM_CRYPTO_TEST_ALG_CF	OFF	Disable CFB mode test
TFM_CRYPTO_TEST_ALG_EC	OFF	Disable ECB mode test
TFM_CRYPTO_TEST_ALG_CT	OFF	Disable CTR mode test
TFM_CRYPTO_TEST_ALG_OF	OFF	Disable OFB mode test
TFM_CRYPTO_TEST_ALG_GC	OFF	Disable GCM mode test
TFM_CRYPTO_TEST_ALG_SH	OFF	Disable SHA-384 algorithm test
TFM_CRYPTO_TEST_ALG_SH	OFF	Disable SHA-512 algorithm test
TFM_CRYPTO_TEST_HKDF	OFF	Disable HKDF algorithm test
TFM_CRYPTO_TEST_ECDH	ON	Enable ECDH key agreement test
TFM_CRYPTO_TEST_CHACHA	OFF	Disable ChaCha20 stream cipher test
TFM_CRYPTO_TEST_CHACHA	OFF	Disable ChaCha20-Poly1305 AEAD algorithm test
TFM_CRYPTO_TEST_SINGLE_	OFF	Test single-part operations in hash, MAC, AEAD and symmetric ciphers

Device configuration extension

To change default configurations and add platform specific configurations, a platform can add a platform configuration file at platform/ext<TFM_PLATFORM>/config.cmake

Crypto service configurations

Crypto Secure Partition

TF-M Profile Medium enables Crypto SP in top-level CMake config file. The following PSA Crypto operationts are enabled by default.

- Hash operations
- Message authentication codes
- Symmetric ciphers
- AEAD operations

[1] PS service is enabled by default. Platforms without off-chip storage devices can turn off TFM_PARTITION_PROTECTED_STORAGE to disable PS service. See *Protected Storage Secure Partition* for details.

- Asymmetric key algorithm based signature and verification
- Key derivation
- Key management

Mbed Crypto configurations

TF-M Profile Medium adds a dedicated Mbed Crypto config file tfm_mbedcrypto_config_profile_medium.h and Mbed Crypto PSA config file crypto_config_profile_medium.h at /lib/ext/mbedcrypto/mbedcrypto_config folder, instead of the common one tfm_mbedcrypto_config_default.h and crypto_config_default.h [CRYPTO-DESIGN].

Major Mbed Crypto configurations are set as listed below:

- Enable SHA256
- Enable generic message digest wrappers
- Enable AES
- Enable CCM mode for symmetric ciphers
- Disable other modes for symmetric ciphers
- Enable ECDH
- Enable ECDSA
- Select ECC curve secp256r1
- Other configurations required by selected option above

Other configurations can be selected to optimize the memory footprint of Crypto module.

A device/use case can append an extra config header to the Profile Medium default Mbed Crypto config file. This can be done by setting the TFM_MBEDCRYPTO_PLATFORM_EXTRA_CONFIG_PATH cmake variable in the platform config file platform/ext<TFM_PLATFORM>/config.cmake. This cmake variable is a wrapper around the MBEDTLS_USER_CONFIG_FILE options, but is preferred as it keeps all configuration in cmake.

Internal Trusted Storage configurations

ITS service is enabled in top-level Profile Medium CMake config file by default.

The internal transient buffer size ITS_BUF_SIZE [ITS-INTEGRATE] is set to 32 bytes by default. A platform/use case can overwrite the buffer size in its specific configuration extension according to its actual requirement of assets and Flash attributes.

Profile Medium CMake config file won't touch the configurations of device specific Flash hardware attributes [ITS-INTEGRATE].

Protected Storage Secure Partition

Data confidentiality, integrity and anti-rollback protection are enabled by default in PS.

If PS is selected, AES-CCM is used as AEAD algorithm by default. It requires to enable PS implementation to select diverse AEAD algorithm.

If platforms don't integrate any off-chip storage device, platforms can disable PS in platform specific configuration extension file via platform/ext<TFM_PLATFORM>/config.cmake.

BL2 setting

Profile Medium enables MCUBoot provided by TF-M by default. A platform can overwrite this configuration by disabling MCUBoot in its configuration extension file platform/ext<TFM_PLATFORM>/config.cmake.

If MCUBoot provided by TF-M is enabled, multiple image boot is selected by default in TF-M Profile Medium top-level CMake config file.

If a device implements its own boot loader, the configurations are implementation defined.

Platform support

To enable Profile Medium on a platform, the platform specific CMake file should be added into the platform support list in top-level Profile Medium CMake config file.

Building Profile Medium

To build Profile Medium, argument TFM_PROFILE in build command line should be set to profile_medium.

Take AN521 as an example:

The following commands build Profile Medium without test cases on **AN521** with build type **MinSizeRel**, built by **Armclang**.

```
cd <TFM root dir>
mkdir build && cd build
cmake -DTFM_PLATFORM=arm/mps2/an521 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTFM_PROFILE=profile_medium \
      -DCMAKE_BUILD_TYPE=MinSizeRel \
      ../
cmake --build ./ -- install
```

The following commands build Profile Medium with regression test cases on **AN521** with build type **MinSizeRel**, built by **Armclang**.

```
cd <TFM root dir>
mkdir build && cd build
cmake -DTFM_PLATFORM=arm/mps2/an521 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTFM_PROFILE=profile_medium \
      -DCMAKE_BUILD_TYPE=MinSizeRel \
      -DTEST_S=ON -DTEST_NS=ON \
      ../
cmake --build ./ -- install
```

Note:

- For devices with more contrained memory and flash requirements, it is possible to build with either only TEST_S enabled or only TEST_NS enabled. This will decrease the size of the test images. Note that both test suites must still be run to ensure correct operation.

More details of building instructions and parameters can be found TF-M build instruction guide [TFM-BUILD].

Reference

Copyright (c) 2020-2022, Arm Limited. All rights reserved.

Trusted Firmware-M Profile Large Design

Introduction

As one of TF-M Profiles, Profile Large protects less resource-constrained Arm Cortex-M devices.

Compared to Profile Small[1] and Profile Medium[2], Profile Large aims to enable more secure features to support higher level of security required in more complex usage scenarios.

- Isolation level 3 enables additional isolation between *Application RoT* (App RoT) services.
- More crypto algorithms and cipher suites are selected to securely connect devices to remote services offered by various major Cloud Service Providers (CSP)
- Basic software countermeasures against physical attacks can be enabled.

Profile Large can be aligned as a reference implementation with the requirements defined in PSA Certified Level 3 Lightweight Protection Profile[3].

Overall design

TF-M Profile Large defines the following feature set:

- Firmware Framework
  - Inter-Process Communication (IPC) model[4]
  - Isolation level 3[Page 1228, 4]
- Internal Trusted Storage (ITS)
- Crypto
  - Support both symmetric ciphers and asymmetric ciphers
  - Asymmetric key based cipher suites defined in TLS 1.2[5] to support direct secure connection to major CSPs, including
    - Authenticated Encryption with Associated Data (AEAD) algorithm
    - Asymmetric key algorithm based signature and verification
    - Public-key cryptography based key exchange
    - Hash function
    - HMAC for default Pseudorandom Function (PRF)
  - Asymmetric digital signature and verification for Initial Attestation Token (IAT)
  - Asymmetric algorithms for firmware image signature verification
  - Key derivation
- Initial Attestation
  - Asymmetric key algorithm based Initial Attestation
- **Secure boot**
  - Anti-rollback protection
  - Multiple image boot
- Protected Storage (PS) if off-chip storage device is integrated
  - Data confidentiality
  - Data integrity
  - Rollback protection

[1] *Trusted Firmware-M Profile Small Design*
[2] *Trusted Firmware-M Profile Medium Design*
[3] PSA Certified Level 3 Lightweight Protection Profile
[4] Arm Platform Security Architecture Firmware Framework 1.0
[5] The Transport Layer Security (TLS) Protocol Version 1.2

- Software countermeasures against physical attacks

Design details

More details of TF-M Profile Large design are described in following sections.

Firmware framework

Profile Large selects IPC model and isolation level 3 by default.

Isolation level 3 supports additional isolation between App RoT services, compared to isolation level 2. It can protect *RoT* services from each other when their vendors don't trust each other.

Crypto service

Profile Large supports direct connection to Cloud services via common protocols, such as TLS 1.2.

In some usage scenarios, PSA RoT can be managed by device manufacturer or other vendors and is out of control of application developers. Profile Large selects alternative crypto algorithms for each crypto function to support multiple common cipher suites required by various major CSPs. Therefore, application developers can support services for diverse CSPs on same devices with Profile Large, without relying on PSA RoT upgrades of crypto.

Devices meeting Profile Large should be in a position to offer at least two alternatives to every cryptographic primitive for symmetric, asymmetric and hash, and be able to use them for encryption, AEAD, signature and verification.

It will cost more resource in Profile Large to support more crypto algorithms and cipher suites, compared to Profile Medium[2].

Boot loader

BL2 implementation can be device specific. Devices may implement diverse boot processes with different features and configurations. However, the boot loader must support anti-rollback protection. Boot loader must be able to prevent unauthorized rollback, to protect devices from being downgraded to earlier versions with known vulnerabilities.

MCUBoot in TF-M is configured as multiple image boot by default in Profile Large. In multiple image boot, secure and non-secure images can be signed independently with different keys and they can be updated separately. It can support multiple vendors scenarios, in which non-secure and secure images are generated and updated by different vendors. Multiple image boot may cost larger memory footprint compared with single image boot.

Boot loader can implement software countermeasures to mitigate physical attacks.

Protected Storage

PS service is required if an off-chip storage device is integrated and used on the platform.

Anti-rollback protection in PS relies on non-volatile counter(s) provided by TF-M Platform *Secure Partition* (SP).

Software countermeasures against physical attacks

TF-M Profile Large enables TF-M Fault Injection Hardening (FIH) library Profile Medium by default. It enables the following countermeasure techniques:

- Control flow monitor
- Failure loop hardening
- Complex constants

- Redundant variables and condition checks

Refer to TF-M physical attack mitigation design document[6] for FIH library details.

Note: TF-M FIH library is still under development.

TF-M FIH library hardens TF-M critical execution steps to make physical attacks more difficult, together with device hardware countermeasures. It is not guaranteed that TF-M FIH library is able to mitigate all kinds of physical attacks.

Note: Implementation note

TF-M FIH library doesn't cover platform specific critical configurations. Platforms shall implement software countermeasures against physical attacks to protect platform specific implementation.

Implementation

Overview

The basic idea is to add dedicated profile CMake configuration files under folder config/profile for TF-M Profile Large default configuration, the same as other TF-M Profiles do.

The top-level Profile Large config file collects all the necessary configuration flags and set them to default values, to explicitly enable the features required in Profile Large and disable the unnecessary ones, during TF-M build.

A platform/use case can provide a configuration extension file to overwrite Profile Large default setting and append other configurations. This configuration extension file can be added via parameter TFM_EXTRA_CONFIG_PATH in build command line.

The behaviour of the Profile Large build flow (particularly the order of configuration loading and overriding) can be found at *Build configuration*

The details of configurations will be covered in each module in *Implementation details*.

Implementation details

This section discusses the details of Profile Large implementation.

Top-level configuration files

The firmware framework configurations in config/profile/profile_large are shown below.

[6] *Physical attack mitigation in Trusted Firmware-M*

Table 34: Config flags in Profile Large top-level CMake config file

Configs	Descriptions	Default value
TFM_ISOLATION_LEVEL	Select level 3 isolation	3
TFM_PARTITION_INTER	Enable ITS SP	ON
TFM_PARTITION_CRYPT	Enable Crypto service	ON
TFM_MBEDCRYPTO_CO	MbedTLS config file path	${CMAKE_SOURCE_DIR}/lib/ext/mbedcrypto/mbedcrypto_config/tfm_mbedcrypto_config_profile_large.h
TFM_MBEDCRYPTO_PS/	MbedTLS PSA config file path	${CMAKE_SOURCE_DIR}/lib/ext/mbedcrypto/mbedcrypto_config/crypto_config_profile_large.h
TFM_PARTITION_INITIA	Enable Initial Attestation service	ON
TFM_PARTITION_PROTI [a]	Enable PS service	ON
TFM_PARTITION_PLATF	Enable TF-M Platform SP	ON

Crypto service configurations

Crypto Secure Partition

TF-M Profile Large enables Crypto SP in top-level CMake config file and selects all the Crypto modules.

MbedTLS configurations

TF-M Profile Large adds a dedicated MbedTLS config file tfm_mbedcrypto_config_profile_large.h and MbedTLS PSA config file crypto_config_profile_large.h under /lib/ext/mbedcrypto/mbedcrypto_config folder, instead of the common one tfm_mbedcrypto_config_default.h and crypto_config_default.h[7].

Major MbedTLS configurations are set as listed below:

- Enable SHA256, SHA384 and SHA512
- Enable generic message digest wrappers
- Enable AES
- Enable CCM mode, GCM mode, CTR mode, CFB mode and CBC mode for symmetric ciphers
- Disable other modes for symmetric ciphers
- Enable ECDH
- Enable ECDSA
- Enable RSA
- Select ECC curve secp256r1 and secp384r1
- Enable HMAC-based key derivation function
- Other configurations required by selected option above

[7] *Crypto design*

A device/use case can append an extra config header to the Profile Large default MbedTLS config file to override the default settings. This can be done by setting the TFM_MBEDCRYPTO_PLATFORM_EXTRA_CONFIG_PATH cmake variable in the platform config file platform/ext<TFM_PLATFORM>/config.cmake. This cmake variable is a wrapper around the MBEDTLS_USER_CONFIG_FILE options, but is preferred as it keeps all configuration in cmake.

Internal Trusted Storage configurations

ITS service is enabled in top-level Profile Large CMake config file by default.

The internal transient buffer size ITS_BUF_SIZE[8] is set to 64 bytes by default. A platform/use case can overwrite the buffer size in its specific configuration extension according to its actual requirement of assets and Flash attributes.

Profile Large CMake config file won't touch the configurations of device specific Flash hardware attributes.

Protected Storage Secure Partition

Data confidentiality, integrity and anti-rollback protection are enabled by default in PS.

If PS is selected, AES-CCM is used as AEAD algorithm by default. If platform hardware crypto accelerator supports the AEAD algorithm, the AEAD operations can be executed in hardware crypto accelerator.

If platforms don't integrate any off-chip storage device, platforms can disable PS in platform specific configuration extension file via platform/ext<TFM_PLATFORM>/config.cmake.

BL2 setting

Profile Large enables MCUBoot provided by TF-M by default. A platform can overwrite this configuration by disabling MCUBoot in its configuration extension file platform/ext<TFM_PLATFORM>/config.cmake.

If MCUBoot provided by TF-M is enabled, multiple image boot is selected by default.

If a device implements its own boot loader, the configurations are implementation defined.

Software countermeasure against physical attacks

Profile Large selects TF-M FIH library Profile Medium by specifying -DTFM_FIH_PROFILE=MEDIUM in top-level CMake config file.

System integrators shall implement software countermeasures in platform specific implementations.

Device configuration extension

To change default configurations and add platform specific configurations, a platform can add a platform configuration file at platform/ext<TFM_PLATFORM>/config.cmake

Test configuration

Some cryptography tests are disabled due to the reduced MbedTLS config. Profile Large specific test configurations are also specified in Profile Large top-level CMake config file config/profile/profile_large_test.cmake.

[8] *ITS integration guide*

Table 35: Profile Large crypto test configuration

Configs	Default value	Descriptions
TFM_CRYPTO_TEST_ALG_CB	ON	Test CBC cryptography mode
TFM_CRYPTO_TEST_ALG_CC	ON	Test CCM cryptography mode
TFM_CRYPTO_TEST_ALG_CF	OFF	Test CFB cryptography mode
TFM_CRYPTO_TEST_ALG_EC	OFF	Test ECB cryptography mode
TFM_CRYPTO_TEST_ALG_CT	OFF	Test CTR cryptography mode
TFM_CRYPTO_TEST_ALG_OF	OFF	Test OFB cryptography mode
TFM_CRYPTO_TEST_ALG_GC	ON	Test GCM cryptography mode
TFM_CRYPTO_TEST_ALG_SH	OFF	Test SHA-384 cryptography algorithm
TFM_CRYPTO_TEST_ALG_SH	ON	Test SHA-512 cryptography algorithm
TFM_CRYPTO_TEST_HKDF	ON	Test HMAC-based key derivation function
TFM_CRYPTO_TEST_ECDH	ON	Test ECDH key agreement algorithm
TFM_CRYPTO_TEST_CHACHA	OFF	Test ChaCha20 stream cipher
TFM_CRYPTO_TEST_CHACHA	OFF	Test ChaCha20-Poly1305 AEAD algorithm
TFM_CRYPTO_TEST_SINGLE_	OFF	Test single-part operations in hash, MAC, AEAD and symmetric ciphers

Platform support

To enable Profile Large on a platform, the platform specific CMake file should be added into the platform support list in top-level Profile Large CMake config file.

Building Profile Large

To build Profile Large, argument TFM_PROFILE in build command line should be set to profile_large.

Take AN521 as an example:

The following commands build Profile Large without test cases on **AN521** with build type **MinSizeRel**, built by **Armclang**.

```
cd <TFM root dir>
mkdir build && cd build
cmake -DTFM_PLATFORM=arm/mps2/an521 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTFM_PROFILE=profile_large \
      -DCMAKE_BUILD_TYPE=MinSizeRel \
      ../
cmake --build ./ -- install
```

The following commands build Profile Large with regression test cases on **AN521** with build type **MinSizeRel**, built by **Armclang**.

```
cd <TFM root dir>
mkdir build && cd build
cmake -DTFM_PLATFORM=arm/mps2/an521 \
      -DTFM_TOOLCHAIN_FILE=../toolchain_ARMCLANG.cmake \
      -DTFM_PROFILE=profile_large \
      -DCMAKE_BUILD_TYPE=MinSizeRel \
      -DTEST_S=ON -DTEST_NS=ON \
      ../
cmake --build ./ -- install
```

More details of building instructions and parameters can be found TF-M build instruction guide[9].

[9] *TF-M build instruction*

Reference

Copyright (c) 2021-2022, Arm Limited. All rights reserved.

Option	Base	Small	ARoT-less	Medium	Large
TFM_ISOLATION_LEVEL	1	1	1	2	3
CONFIG_TFM_SPM_BACKEND	SFN	SFN	SFN	IPC	IPC
TFM_PARTITION_CRYPTO	OFF	ON	ON	ON	ON
TFM_PARTITION_INTERNAL_TRUSTED_S	OFF	ON	ON	ON	ON
TFM_PARTITION_PLATFORM	OFF	OFF	ON	ON	ON
TFM_PARTITION_PROTECTED_STORAGE	OFF	OFF	OFF	ON	ON
TFM_PARTITION_INITIAL_ATTESTATION	OFF	ON	ON	ON	ON
SYMMETRIC_INITIAL_ATTESTATION	OFF	ON	OFF	OFF	OFF
TFM_PARTITION_FIRMWARE_UPDATE	OFF	OFF	ON	OFF	OFF
PS_CRYPTO_AEAD_ALG	GCM	-	-	CCM	CCM
PSA_FRAMEWORK_HAS_MM_IOVEC	OFF	ON	OFF	OFF	OFF
MCUBOOT_IMAGE_NUMBER [1]	2	1	2	2	2
Advanced options, defined in the corresponded header (.h) file					
CRYPTO_ENGINE_BUF_SIZE	0x2080	0x400	0x2080	0x2080	0x2380
CRYPTO_ASYM_SIGN_MODULE_ENABLEI	ON	OFF	ON	ON	ON
CRYPTO_ASYM_ENCRYPT_MODULE_ENA	ON	OFF	OFF	OFF	ON
CRYPTO_SINGLE_PART_FUNCS_DISABLEI	OFF	ON	OFF	OFF	OFF
CRYPTO_CONC_OPER_NUM	8	4	8	8	8
CONFIG_TFM_CONN_HANDLE_MAX_NUI	8	3	8	8	8
ITS_BUF_SIZE [2]	512	32	32	32	512

1. *MCUBOOT_IMAGGE_NUMBER* value is taken from MCUBoot default configuration, execept profile Small.
2. Many platforms redefine *ITS_BUF_SIZE* value.

Each profile has predefined configuration for cryptographic library, located in /lib/ext/mbedcrypto/mbedcrypto_config/

Copyright (c) 2020, Arm Limited. All rights reserved.

Tests configuration

Regression tests

Regression test configuration is controlled entirely by TEST_NS or TEST_S or single test suite configuration. The group test configurations and single test suite configurations are listed below, all of them are disabled by default.

Parameter	Description
TEST_NS_ATTESTATION	Build non-secure regression Attestation tests.
TEST_NS_T_COSE	Build non-secure regression t_cose tests.
TEST_NS_QCBOR	Build non-secure regression QCBOR tests.
TEST_NS_CRYPTO	Build non-secure regression Crypto tests.
TEST_NS_ITS	Build non-secure regression ITS tests.
TEST_NS_PS	Build non-secure regression PS tests.
TEST_NS_PLATFORM	Build non-secure regression Platform tests.
TEST_NS_FWU	Build non-secure regression FWU tests.
TEST_NS_IPC	Build non-secure regression IPC tests.
TEST_NS_SLIH_IRQ	Build non-secure regression Second-Level Interrupt Handling tests.
TEST_NS_FLIH_IRQ	Build non-secure regression First-Level Interrupt Handling tests.
TEST_NS_MULTI_CORE	Build non-secure regression multi-core tests.
TEST_S_ATTESTATION	Build secure regression Attestation tests.
TEST_S_CRYPTO	Build secure regression Crypto tests.
TEST_S_ITS	Build secure regression ITS tests.
TEST_S_PS	Build secure regression PS tests.
TEST_S_PLATFORM	Build secure regression Platform tests.
TEST_S_FWU	Build secure regression FWU tests.
TEST_S_IPC	Build secure regression IPC tests.

Individual test suites can be enabled when their dependencies like partitions or other specific configurations are set. On the one hand, some test suites depend on other test suites. On the other hand, some test suites conflict with other test suites. Test configurations and dependencies will be checked in ${TFM_TEST_REPO_PATH}/test/config/check_config.cmake.

If regression testing is enabled by TEST_NS or TEST_S, individual test suites will be enabled or disabled as appropriate for the TF-M configuration (i.e. all enabled secure partitions will be tested).

Some cryptographic tests can be enabled and disabled. This is done to prevent false failures from being reported when a smaller Mbed Crypto config is being used which does not support all features.

Parameter	Description	Default value
TFM_CRYPTO_TEST_ALG_CBC	Test CBC cryptography mode	ON
TFM_CRYPTO_TEST_ALG_CCM	Test CCM cryptography mode	ON
TFM_CRYPTO_TEST_ALG_CFB	Test CFB cryptography mode	ON
TFM_CRYPTO_TEST_ALG_ECB	Test ECB cryptography mode	ON
TFM_CRYPTO_TEST_ALG_CTR	Test CTR cryptography mode	ON
TFM_CRYPTO_TEST_ALG_OFB	Test OFB cryptography mode	ON
TFM_CRYPTO_TEST_ALG_GCM	Test GCM cryptography mode	ON
TFM_CRYPTO_TEST_ALG_SHA_384	Test SHA-384 cryptography algorithm	ON
TFM_CRYPTO_TEST_ALG_SHA_512	Test SHA-512 cryptography algorithm	ON
TFM_CRYPTO_TEST_HKDF	Test HKDF key derivation algorithm	ON
TFM_CRYPTO_TEST_ECDH	Test ECDH key agreement algorithm	ON
TFM_CRYPTO_TEST_CHACHA20	Test ChaCha20 stream cipher	ON
TFM_CRYPTO_TEST_ALG_CHACHA20_POLY	Test ChaCha20-Poly1305 AEAD algorithm	ON

PSA tests

PSA tests are configured by using the TEST_PSA_API cmake variable. The variable should be set to the name of the test suite that is desired. It is _not_ supported to set both TEST_PSA_API and TEST_S or TEST_NS.

The Functional API tests are:

- CRYPTO
- INITIAL_ATTESTATION
- STORAGE (INTERNAL_TRUSTED_STORAGE and PROTECTED_STORAGE)
- INTERNAL_TRUSTED_STORAGE
- PROTECTED_STORAGE

The Firmware Framework test suites are:

- IPC

Note that these map directly to the SUITE cmake variable used in the psa-arch-tests documentation.

TF-M is highly configurable project with many configuration options to meet a user's needs. A user can select the desired set of services and fine-tune them to their requirements. There are two types of configuration options

Build configuration
Specifies which file or component to include into compilation and build. These are options, usually used by a build system to enable/disable modules, specify location of external dependency or other selection, global to a project. These option set shall be considered while adopting TF-M to other build systems. In the *Base Configuration* table these options have *Build* type.

Component configuration
To adjust a particular parameter to a desired value. Those options are local to a component or externally referenced when components are coupled. Options are in C header file. The *The Header File Config System* has more details about it. In the *Base Configuration* table these options have *Component* type.

Note: Originally, TF-M used CMake variables for both building and component tuning purposes. It was convenient to have a single system for both building and component's configurations. To simplify and improve configurability and better support build systems other than a CMake, TF-M introduced a *The Header File Config System* and moved component options into a dedicated config headers.

How to configure

TF-M Project provides a base build, defined in /config/config_base.cmake and /config/config_base.h. Starting from the base, users can enable required services and features using several independent methods to configure TF-M.

Use *TF-M Profiles*.
There are 4 sets of predefined configurations for a elected use cases, called profiles. A user can select a profile by providing -DTFM_PROFILE=<profile file name>. Each profiles represented by a pair of configuration files for Building (CMake) options and Component options (.h file)

Use a custom profile.
Another method is to take an existing TF-M profile and adjust the desired options manually editing CMake and config header files. This is for users familiar with TF-M.

Use *The Kconfig System*.
This method is recommended for beginners. Starting from the *base configuration* a user

can enable necessary services and options. KConfig ensurers that all selected options are consistent and valid. This is new in v1.7.0 and it covers only SPM and PSA services. As an output KConfig produces a pair of configuration files, similar to a profile.

Note: In contrast, before TF-M v1.7.0, the default build includes all possible features. With growing functionality, such rich default build became unpractical by not fitting into every platform and confusing of big memory requirements.

Priorities

A project configuration performed in multiple steps with priorities. The list below explains the process but for the details specific to *Build configuration* or *The Header File Config System* please check the corresponded document.

1. The base configuration with default values is used as a starting point
2. A profile options applied on top of the base
3. A platform can check the selected configuration and apply restrictions
4. Finally, command line options can modify the composed set

Note: To ensure a clear intention and conscious choice, all options must be provided explicitly via a project configuration file. Default values on step 1 will generate warnings which are expected to break a build.

Base Configuration

The base configuration is the ground for configuring TF-M, provided defaults are defined in /config/config_base.cmake and /config/config_base.h. The base build includes SPM and platform code only.

This table lists the config option categorizations of the SPM and Secure Partitions.

Crypto

Options	Type	Base Value
TFM_PARTITION_CRYPTO	Build	OFF
CRYPTO_TFM_BUILTIN_KEYS_DRIVER	Build	ON
CRYPTO_NV_SEED	Component	ON
CRYPTO_ENGINE_BUF_SIZE	Component	0x2080
CRYPTO_IOVEC_BUFFER_SIZE	Component	5120
CRYPTO_STACK_SIZE	Component	0x1B00
CRYPTO_CONC_OPER_NUM	Component	8
CRYPTO_RNG_MODULE_ENABLED	Component	1
CRYPTO_KEY_MODULE_ENABLED	Component	1
CRYPTO_AEAD_MODULE_ENABLED	Component	1
CRYPTO_MAC_MODULE_ENABLED	Component	1
CRYPTO_HASH_MODULE_ENABLED	Component	1
CRYPTO_CIPHER_MODULE_ENABLED	Component	1
CRYPTO_ASYM_SIGN_MODULE_ENABLED	Component	1
CRYPTO_ASYM_ENCRYPT_MODULE_ENABLED	Component	1
CRYPTO_KEY_DERIVATION_MODULE_ENABLED	Component	1
CRYPTO_SINGLE_PART_FUNCS_ENABLED	Component	1

Initial Attestation

Options	Type	Base Value
TFM_PARTITION_INITIAL_ATTESTATION	Build	OFF
SYMMETRIC_INITIAL_ATTESTATION	Build	OFF
ATTEST_INCLUDE_TEST_CODE	Build	OFF
ATTEST_KEY_BITS	Build	256
ATTEST_TOKEN_PROFILE	Component	“PSA_IOT_1”
ATTEST_INCLUDE_OPTIONAL_CLAIMS	Component	1
ATTEST_INCLUDE_COSE_KEY_ID	Component	0
ATTEST_STACK_SIZE	Component	0x700

Internal Trusted Storage

Options	Type	Base Value
TFM_PARTITION_INTERNAL_TRUSTED_STORAGE	Build	OFF
ITS_CREATE_FLASH_LAYOUT	Component	1
ITS_RAM_FS	Component	0
ITS_VALIDATE_METADATA_FROM_FLASH	Component	1
ITS_MAX_ASSET_SIZE	Component	512
ITS_NUM_ASSETS	Component	10
ITS_BUF_SIZE	Component	ITS_MAX_ASSET_SIZE
ITS_STACK_SIZE	Component	0x720

Protected Storage

Options	Type	Base Value
TFM_PARTITION_PROTECTED_STORAGE	Build	OFF
PS_ENCRYPTION	Build	ON
PS_CRYPTO_AEAD_ALG	Build	PSA_ALG_GCM
PS_CREATE_FLASH_LAYOUT	Component	1
PS_RAM_FS	Component	0
PS_VALIDATE_METADATA_FROM_FLASH	Component	1
PS_MAX_ASSET_SIZE	Component	2048
PS_NUM_ASSETS	Component	10
PS_ROLLBACK_PROTECTION	Component	1
PS_STACK_SIZE	Component	0x700

Firmware Update

Options	Type	Base Value
PLATFORM_HAS_FIRMWARE_UPDATE_SU	Build	OFF
TFM_PARTITION_FIRMWARE_UPDATE	Build	OFF
TFM_CONFIG_FWU_MAX_WRITE_SIZE	Build	1024
TFM_CONFIG_FWU_MAX_MANIFEST_SIZI	Build	0
FWU_DEVICE_CONFIG_FILE	Build	“”
FWU_SUPPORT_TRIAL_STATE	Build	Depends on MCUBOOT_UPGRADE_STRATEGY
TFM_FWU_BOOTLOADER_LIB	Build	“mcuboot”
TFM_FWU_BUF_SIZE	Component	PSA_FWU_MAX_BLOCK_SIZE
FWU_STACK_SIZE	Component	0x600

Platform Secure Partition

Options	Type	Base Value
TFM_PARTITION_PLATFORM	Build	OFF
PLATFORM_SERVICE_INPUT_BUFFER_SIZE	Component	64
PLATFORM_SERVICE_OUTPUT_BUFFER_SIZE	Component	64
PLATFORM_SP_STACK_SIZE	Component	0x500
PLATFORM_NV_COUNTER_MODULE_DISABLED	Component	0

NS Agent Mailbox Secure Partition

Options	Type	Base Value
NS_AGENT_MAILBOX_STACK_SIZE	Component	0x800

Secure Partition Manager

Options	Type	Base Values
TFM_ISOLATION_LEVEL	Build	1
PSA_FRAMEWORK_HAS_MM_IOVEC	Build	OFF
CONFIG_TFM_SPM_BACKEND	Build	“SFN”
TFM_SPM_LOG_LEVEL	Build	1
CONFIG_TFM_CONN_HANDLE_MAX_NUM	Component	8
CONFIG_TFM_DOORBELL_API	Component	0
CONFIG_TFM_SCHEDULE_WHEN_NS_INTERRUPTED	Component	0

Copyright (c) 2022,2024, Arm Limited. All rights reserved. Copyright (c) 2023 Cypress Semiconductor Corporation (an Infineon company) or an affiliate of Cypress Semiconductor Corporation. All rights reserved.

Integration Guide

The purpose of this document is to provide a guide on how to integrate TF-M with other hardware platforms and operating systems.

Source Structure

TF-M is designed to provide a user-friendly source structure to ease integration and service development. This document introduces the source structure and its design intention of TF-M. Additionally, the folders below are important for TF-M integration and descibed in a separate documents:

Details for the platform folder

Interfacing with TF-M

platform/ext/target/tfm_peripherals_def.h

This file should enumerate the hardware peripherals that are available for TF-M on the platform. The name of the peripheral used by a service is set in its manifest file. The platform have to provide a macro for each of the provided peripherals, that is substituted to a pointer to type `struct platform_data_t`. The memory that the pointer points to is allocated by the platform code.

The pointer gets stored in the partitions database in SPM, and it is provided to the SPM HAL functions.

Peripherals currently used by the services in TF-M

- TFM_PERIPHERAL_UART1- UART1

Note: If a platform doesn't support a peripheral, that is used by a service, then the service cannot be used on the given platform. Using a peripheral in TF-M that is not supported by the platform results in compile error.

---

platform/include/tfm_spm_hal.h

This file contains the declarations of functions that a platform implementation has to provide for TF-M's SPM. For details see the comments in the file.

platform/include/tfm_platform_system.h

This file contains the declarations of functions that a platform implementation has to provide for TF-M's Platform Service. For details see docs/user_guides/services/tfm_platform_integration_guide.rst

System Startup

Before calling main(), platform startup code should initialise all system clocks, perform runtime initialisation and other steps such as setting the vector table. Configuration of the following features is optional as TF-M architecture code will check and initialise them:

- The Security Extension.
- The Floating-point Extension.

Debug authentication settings

A platform may provide the option to configure debug authentication. TF-M core calls the HAL function enum tfm_hal_status_t tfm_hal_platform_init(void) in which debug authentication is configured based on the following defines:

- *DAUTH_NONE*: Debugging the system is not enabled.
- *DAUTH_NS_ONLY*: Invasive and non invasive debugging of non-secure code is enabled.
- *DAUTH_FULL*: Invasive and non-invasive debugging of non-secure and secure code is enabled.
- *DAUTH_CHIP_DEFAULT*: The debug authentication options are used that are set by the chip vendor.

The desired debug authentication configuration can be selected by setting one of the options above to the cmake command with the *-DDEBUG_AUTHENTICATION="<define>"* option. The default value of *DEBUG_AUTHENTICATION* is *DAUTH_CHIP_DEFAULT*.

Note: enum tfm_hal_status_t tfm_hal_platform_init(void) is called during the TF-M core initialisation phase, before initialising secure partition. This means that BL2 runs with the chip default setting.

Sub-folders

include

This folder contains the interfaces that TF-M expects every target to provide. The code in this folder is created as a part of the TF-M project therefore it adheres to the BSD 3.0 license.

ext

This folder contains code that has been imported from other projects so it may have licenses other than the BSD 3.0 used by the TF-M project.

Please see the *Details for the platform/ext folder* for details.

Copyright (c) 2017-2023, Arm Limited. All rights reserved.

Details for the platform/ext folder

This folder has code that has been imported from other projects. This means the files in this folder and subfolders have Apache 2.0 license which is different to BSD 3.0 license applied to the parent TF-M project.

Note: This folder is strictly Apache 2.0 with the exception of cmake files. Maintainers should be consulted if this needs to be revisited.

Sub-folders

accelerator

This folder contains cmake and code files to interact cryptographic accelerators.

In order to use a cryptographic accelerator, a platform must set CRYPTO_HW_ACCELERATOR_TYPE in preload.cmake. This option maps directly to the subdirectories of the accelerator directory. Currently available accelerators are : the CryptoCell cc312, the STMicroelectronics accelerator stm .

A minimal API is exposed to interact with accelerators, the details of this api are in accelerator/interface/crypto_hw.h. Implementation of the API is inside the subdirectory of the individual accelerator.

To configure a cryptographic accelerator at build time, two cmake options can be specified.

- CRYPTO_HW_ACCELERATOR
  - ON All possible mbedtls cryptographic operations will be offloaded to the accelerator.
  - OFF The cryptographic accelerator will be ignored and software cryptography will be used.

cmsis

This folder contains core and compiler specific header files imported from the CMSIS_5 project.

common

armclang and gcc

These contain the linker scripts used to configure the memory regions in TF-M regions.

template

This directory contains platform-independent dummy implementations of the interfaces in platform/include. These implementations can be built directly for initial testing of a platform port, or used as a basic template for a real implementation for a particular target. They **must not** be used in production systems.

driver

This folder contains the headers with CMSIS compliant driver definitions that that TF-M project expects a target to provide.

target_cfg.h

This file is expected to define the following macros respectively.

- TFM_DRIVER_STDIO - This macro should expand to a structure of type ARM_DRIVER_USART. TFM redirects its standard input and output to this instance of USART.
- NS_DRIVER_STDIO - This macro should expand to a structure of type ARM_DRIVER_USART. Non-Secure application redirects its standard input and output to this instance of USART.

target

This folder contains the files for individual target. For a buildable target, the directory path from the target directory to its CMakeLists.txt file is the argument that would be given to -DTFM_PLATFORM=.

The standard directory structure is as follows:

- **target**
  - **<Vendor name>**
    - common
    - <buildable target 1>
    - <buildable target 2>
    - <buildable target 3>

Each buildable target must contain the cmake files mandated in the section below.

The common directory is not required, but can be used to contain code that is used by multiple targets.

There must not be any directories inside the vendor directory that is not either the common directory or a buildable platform, to avoid confusion about what directories are a valid TFM_PLATFORM.

Buildable target required cmake files

A buildable target must provide 3 mandatory cmake files. These files must all be placed in the root of the buildable target directory.

preload.cmake

This file contains variable definitions that relate to the underlying hardware of the target.

- TFM_SYSTEM_PROCESSOR: The processor used by the target. The format is that same as the format used in the -mcpu= argument of GNUARM or ARMCLANG. The special +modifier syntax must not be used.

- TFM_SYSTEM_ARCHITECTURE: The architecture used by the target. The format is that same as the format used in the -march= argument of GNUARM or ARMCLANG. The special +modifier syntax must not be used.
- TFM_SYSTEM_DSP: Whether the target has the DSP feature of the given TFM_SYSTEM_PROCESSOR
- CRYPTO_HW_ACCELERATOR_TYPE: The type of cryptographic accelerator the target has, if it has one. This maps exactly to the subdirectories of platform/ext/accelerator
- CONFIG_TFM_FP_ARCH: The "FPU architecture" (Armclang) or "FP hardware" (GNU) used by the target for compiling C source files. The value will be transferred to -mfpu argument of GNUARM or ARMCLANG
- CONFIG_TFM_FP_ARCH_ASM: The "FPU architecture" (Armclang) used by the target for assembling ASM source files. This value will be transferred to the --fpu argument of ARMCLANG. This value is not used by GNUARM.

For more details on CONFIG_TFM_FP_ARCH and CONFIG_TFM_FP_ARCH_ASM, please refer to *Floating-Point Support*.

Other than these particular cmake variables, it is permissible for the preload.cmake file to contain add_definitions statements, in order for set compile definitions that are global for the hardware. This is commonly used to select a particular set of code from a vendor SDK.

It is not permissible to contains code other than the above in a preload.cmake file, any general cmake code should be placed in CMakeLists.txt and any configuration options should be contained in config.cmake

config.cmake

This file collects platform-specific overrides to the configuration options. This should only contain cmake options that are included in config_base.cmake. These options should be set as CACHE variables, as they are in config_base.cmake.

CMakeLists.txt

This file should contain all other required cmake code for the platform. This primarily consists of the following:

- Adding an include directory to the target platform_region_defs, which contains the headers flash_layout.h and region_defs.h
- Adding startup and scatter files to the tfm_s, tfm_ns and bl2 targets.
- linking CMSIS_5_tfm_ns to the correct version of the CMSIS RTX libraries, as defined in lib/ext/CMSIS_5/CMakeLists.txt
- Adding required source files, include directories and compile definitions to the platform_s, platform_ns and platform_bl2 targets.

preload_ns.cmake

This optional cmake file is required only if the target runs the NSPE on a core that requires different compiler options than the SPE core. This file has the same format as preload.cmake, but instead details the hardware of the NS core that is **not** running the main TF-M secure code.

install.cmake

This optional cmake file is required only if additional files need to be installed for the platform.

Flash layout header file

Target must provide a header file, called flash_layout.h, which defines the information explained in the follow subsections. The defines must be named as they are in the subsections.

BL2 bootloader

The BL2 bootloader requires the following definitions:

- FLASH_BASE_ADDRESS - Defines the first valid address in the flash.
- FLASH_AREA_BL2_OFFSET - Defines the offset from the flash base address where the BL2 - MCUBOOT area starts.
- FLASH_AREA_BL2_SIZE - Defines the size of the BL2 area.
- FLASH_AREA_SCRATCH_OFFSET - Defines the offset from the flash base address where the scratch area starts, which is used during image swapping.
- FLASH_AREA_SCRATCH_SIZE - Defines the size of the scratch area. The minimal size must be as the biggest sector size in the flash.
- FLASH_DEV_NAME - Specifies the flash device used by BL2.

The BL2 requires further definitions depending on the number of images, the meaning of these macros are also slightly different:

- Required definitions in case of 1 image (S and NS images are concatenated and handled together as one binary blob):
  - FLASH_AREA_0_OFFSET - Defines the offset from the flash base address where the primary image area starts, which hosts the active firmware image.
  - FLASH_AREA_0_SIZE - Defines the size of the primary image area.
  - FLASH_AREA_2_OFFSET - Defines the offset from the flash base address where the secondary image area starts, which is a placeholder for new firmware images.
  - FLASH_AREA_2_SIZE - Defines the size of the secondary image area.
- Required definitions in case of 2 images (S and NS images are handled and updated separately):
  - FLASH_AREA_0_OFFSET - Defines the offset from the flash base address where the primary image areas start, which host the active firmware images. It is also the offset of the primary (active) secure image area.
  - FLASH_AREA_0_SIZE - Defines the size of the primary secure image area.
  - FLASH_AREA_1_OFFSET - Defines the offset from the flash base address where the primary (active) non-secure image area starts.
  - FLASH_AREA_1_SIZE - Defines the size of the primary non-secure image area.
  - FLASH_AREA_2_OFFSET - Defines the offset from the flash base address where the secondary image areas start, which are placeholders for new firmware images. It is also the offset of the secondary secure image area.
  - FLASH_AREA_2_SIZE - Defines the size of the secondary secure image area.
  - FLASH_AREA_3_OFFSET - Defines the offset from the flash base address where the secondary non-secure image area starts.
  - FLASH_AREA_3_SIZE - Defines the size of the secondary non-secure image area.

The table below shows a fraction of the flash layout in case of 2 and 1 updatable images with the related flash areas that hold the firmware images:

Image number: 2		Image number: 1	
Flash area	**Content**	**Flash area**	**Content**
FLASH_AREA_0	Secure image primary slot	FLASH_AREA_0	Secure + Non-secure image primary slot
FLASH_AREA_1	Non-secure image primary slot		
FLASH_AREA_2	Secure image secondary slot	FLASH_AREA_2	Secure + Non-secure image secondary slot
FLASH_AREA_3	Non-secure image secondary slot		
FLASH_AREA_SCRATCH	Scratch area	FLASH_AREA_SCRATCH	Scratch area

- IMAGE_EXECUTABLE_RAM_START - Defines the start of the region to which images are allowed to be loaded. Only used if MCUBOOT_UPGRADE_STRATEGY is configured to be RAM_LOAD.
- IMAGE_EXECUTABLE_RAM_SIZE - Defines the size of the region to which images are allowed to be loaded. Only used if MCUBOOT_UPGRADE_STRATEGY is configured to be RAM_LOAD.

Assemble tool

The assemble.py tool is used to concatenate secure and non-secure binary to a single binary blob. It requires the following definitions:

- SECURE_IMAGE_OFFSET - Defines the offset from the single binary blob base address, where the secure image starts.
- SECURE_IMAGE_MAX_SIZE - Defines the maximum size of the secure image area.
- NON_SECURE_IMAGE_OFFSET - Defines the offset from the single binary blob base address, where the non-secure image starts.
- NON_SECURE_IMAGE_MAX_SIZE - Defines the maximum size of the non-secure image area.

Image tool

The imgtool.py tool is used to handle the tasks related to signing the binary. It requires the following definition:

- S_IMAGE_LOAD_ADDRESS - Defines the address to where the Secure (or combined Secure and Non-secure) image is loaded and is executed from. Only used if MCUBOOT_UPGRADE_STRATEGY is configured to be RAM_LOAD.
- NS_IMAGE_LOAD_ADDRESS - Defines the address to where the Non-secure image is loaded and is executed from. Only used if MCUBOOT_UPGRADE_STRATEGY is configured to be RAM_LOAD and MCUBOOT_IMAGE_NUMBER is greater than 1.

Expose target support for HW components

Services may require HW components to be supported by the target to enable some features (e.g. PS service with rollback protection, etc). The following definitions need to be set in the .cmake file if the target has the following HW components:

- TARGET_NV_COUNTERS_ENABLE - Specifies that the target has non-volatile (NV) counters.

Copyright (c) 2017-2023, Arm Limited. All rights reserved. Copyright (c) 2020-2022 Cypress Semiconductor Corporation (an Infineon company) or an affiliate of Cypress Semiconductor Corporation. All rights reserved.

/ (TF-M root)

This table describes the structure under the root folder with part of possible folders.

Folder name	Description
bl1	The 1st stage immutable bootloader
bl2	MCUBoot based 2nd stage bootloader
cmake	Cmake files of the build system
config	Configuration system files
docs	The documentation
interface	RoT service API for client calls
lib	The 3rd party libraries
platform	Platform intermedia files
secure_fw	The secure firmware
tools	Tools in scripts for building

platform

The *platform* folder contains SW ports of all *supported platforms* and the files necessary for *adding a new platform*. Please refer to *platform folder document* for more information.

Folder name	Description
include	HAL and platform public headers
ext	Platform ports and related files

platform/ext

This folder can include imported files licensed differently from TF-M's BSD-3 license. More details are in the dedicated document *Details for the platform/ext folder*.

Folder name	Description
accelerator	Supported Crypto HW accelerators
cmsis	A copy of essential CMSIS headers
common	Common HAL implementation
driver	Driver headers for porting
target/<vendor>	Vendor specific folders with ported platforms

Each *vendor* is assigned one folder for usage under *target/*. A *vendor* contributes platform's ports and manages the structure inside it.

secure_fw

This folder contains components needed by secure firmware and the exported interfaces for application, service development and HAL integration.

Folder name	Description
include	Public headers of *secure_fw*
partitions	Default services and SPRTL
spm	PSA FF-M SPM implementation
shared	Sources shared out of SPRTL

The shared sources can be referenced by the building system out of SPRTL. Generally, they are runtime and PSA APIs.

secure_fw/include

This folder holds public headers for external references by clients, services and platforms. Avoid putting private headers, not referenced by other modules in this *include* folder.

secure_fw/partitions

This folder contains default services implemented as partitions and runtime utilities used and provided by TF-M.

Folder name	Description
lib/runtime	The SPRTL sources and intermedia files
<partition_x>	Sources of *partition_x*
<partition_x>/include	RoT Service API headers of *partition_x*
<partition_y>	Sources of *partition_y*
<partition_y>/include	RoT Service API headers of *partition_y*

Here *partition_x* and *partition_y* are examples of RoT services without detailed structure of them.

secure_fw/spm

The SPM is the core component to provide a mechanism for providing secure services complied with PSA FF-M.

Folder name	Description
include	SPM public headers.
core	SPM base functionalities
*ext	Extended SPM functionalities

Copyright (c) 2020-2023, Arm Limited. All rights reserved.

Copyright (c) 2023, Arm Limited. All rights reserved.

SPM Backends

This document briefly introduces the backends of Secure Partition Manager (SPM) in TF-M and how to select one for building.

IPC and SFN

The Firmware Framework M (FF-M)[1],[2] provides two different programming models for Secure Partitions.

- IPC Model

  The Secure Partition processes signals in any order, and can defer responding to a message while continuing to process other signals.

- SFN Model

  The Secure Partition is made up of a collection of callback functions which implement secure services.

Although the programming model is different, they share the same APIs to interact with the SPM. The behaviours of the APIs share the same implementation with slight differences for the two programming models. This is regarded as the frontend.

The TF-M runtime implementations behind frontend are different. TF-M provides two backends correspondingly.

- IPC backend

  In this backend, the SPM and each Secure Partition have their own execution contexts, which is required to support the IPC model Secure Partitions. This also enables the SPM to provide higher isolation levels. This SPM backend acts like a multiple-process system. It can also adopt SFN model Secure Partitions.

- SFN backend

  The SFN backend provides more efficient executions because it shares a single-thread execution context with all the Secure Partitions. This SPM backend acts like a single library. Therefore, it can only adopt SFN model Secure Partitions. And it does not support higher isolation levels. On the other hand, it consumes less memory compared to the IPC backend.

The following table summaries the relationships between SPM backends, Secure Partition models and isolation levels.

SPM backend	Supported Partition model	Supported Isolation Level
SFN	SFN Partition	1
IPC	IPC and SFN Partition	1, 2 and 3

Implementation Recommendations

If an implementation doesn't contain any IPC model Secure Partition and only requires isolation level 1, then it is recommended to select the SFN backend to optimize memory consumption and execution performance.

If an implementation contains any IPC model Secure Partition or requires isolation level 2 or 3, then the IPC backend is required.

TF-M Configuration Switches

In the TF-M build system, the CONFIG_TFM_SPM_BACKEND configuration is used to select the backend of SPM. The valid values are SFN and IPC.

```
-DCONFIG_TFM_SPM_BACKEND=SFN
```

[1] FF-M v1.0 Specification
[2] FF-M v1.1 Extension

If CONFIG_TFM_SPM_BACKEND is not set, then IPC is the default value.

References

Copyright (c) 2022-2023, Arm Limited. All rights reserved.

Non-secure Client Extension Integration Guide

This document introduces TF-M Non-secure Client Extension (NSCE) and how to integrate it with Non-secure Processing Environment (NSPE) RTOS.

What is NSCE for

Besides the secure services provided via PSA APIs, there are two interactions between TF-M and NSPE RTOS.

- Non-secure context management in TF-M

  When a NS task calls a secure service, a context is maintained in TF-M. If TF-M supports multiple secure service calls, the context needs to be loaded and saved with the corresponding NS task when the RTOS kernel (the kernel) does scheduling.

- Non-secure client ID (NSID) management

  As per PSA Firmware Framework specification, NSID is required for a secure service call from the NSPE. The NSID can be managed by either SPM (the same NSID must be used for all connections) or the NSPE RTOS. For the latter case, the NSPE RTOS must provide the NSID for each connection.

NSCE is implemented to support the interactions above.

- Provide the interface to NSPE RTOS for managing the context in TF-M.
- Provide a mechanism to NSPE RTOS for providing the NSID for each connection.

NSCE component diagram

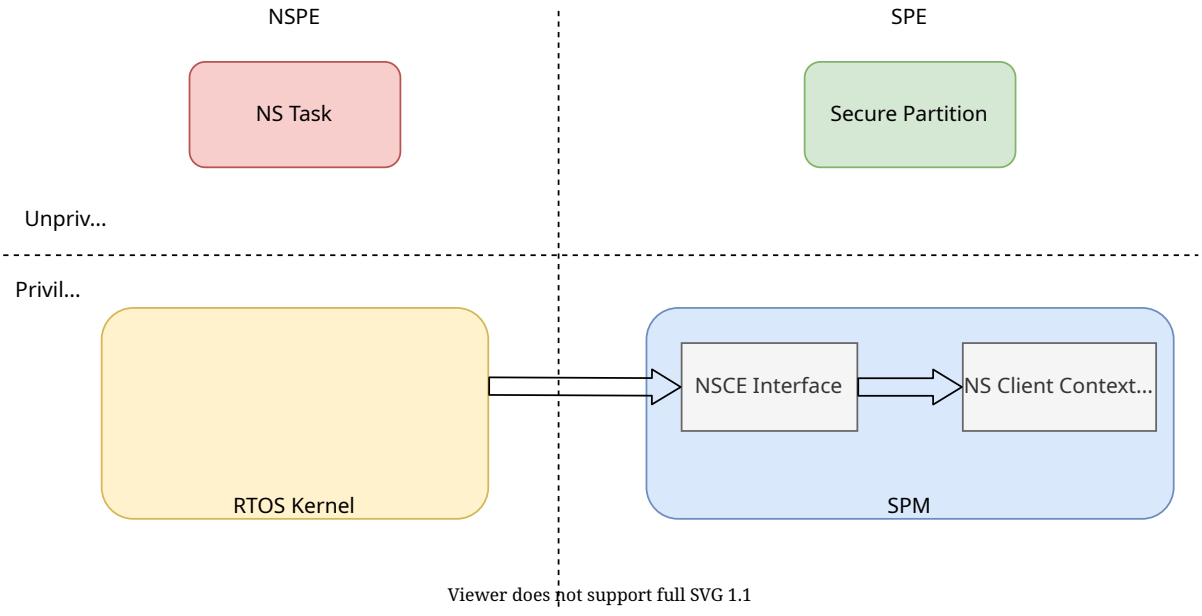


Fig. 2: Non-secure Client Extension Component Diagram

This diagram shows the components of NSCE.

- NSCE interface: The NSCE API interface to NSPE RTOS.
- NS Client Context Manager: It is the internal component of SPM that is used to manage the NS client context. It is not accessible to NSPE.

Group-based context management

NSCE introduces a group-based context management for NS clients. The purpose is to support diverse use scenarios in the IoT world.

For rich-resource devices, TF-M could assign separate contexts for each connection. For resource-constrained devices, TF-M may support very limited context slots. Multiple NS connections could share the same context provided they have mutual exclusion access to the secure services.

To support this flexibility, a group ID (gid) and a thread ID (tid) are specified for the connection of the NS client. NSCE allocates only one context for a given group. Threads in the same group share the context.

The gid and tid are specified by NSPE RTOS via NSCE interface.

NSCE interface

NSCE defines a set of APIs as the interface for NSPE RTOS to manage the context in TF-M. NSID is provided as an input parameter of *tfm_nsce_load_ctx()*.

NSCE APIs are typically called by the kernel and must be called from the non-secure handler mode.

```
uint32_t tfm_nsce_init(uint32_t ctx_requested)
```

This function should be called before any other NSCE APIs below to do non-secure context initialization in NSCE for the calling NS entity. *ctx_requested* is the number of the contexts requested by the NS entity. The number of assigned context will be returned. It may be equal to or smaller than the requested context number based on the current available resource in TF-M. If *ctx_requested* is *0*, then the maximum available context number will be assigned and returned. If the initialization is failed, *0* should be returned.

Note: As TF-M only supports one context for now, so the return value is always *1* if no error. Currently, it is safe to skip calling *tfm_nsce_init()*. But, for future compatibility, it is recommended to do so.

```
uint32_t tfm_nsce_acquire_ctx(uint8_t group_id, uint8_t thread_id)
```

This function allocates a context for the NS client connection. The *gid* and *tid* are the input paramemters. A token will be returned to the NSPE if TF-M has an available context slot. Otherwise, *TFM_NS_CLIENT_INVALID_TOKEN* is returned. It is the responsibility of NSPE RTOS to assign gid and tid for each NS client.

```
uint32_t tfm_nsce_release_ctx(uint32_t token)
```

This function tries to release the context which was retrieved by *tfm_nsce_acquire_ctx()*. As the context may be shared with other threads within the same group, the context is only freed and back to the available context pool after all threads in the same group release the context.

```
uint32_t tfm_nsce_load_ctx(uint32_t token, int32_t nsid)
```

This function should be called when NSPE RTOS schedules in a NS client. *token* is returned by *tfm_nsce_acquire_ctx()*. *nsid* is the non-secure client ID used for the following PSA service calls.

The assignment of NSID is managed by the NSPE RTOS. It is not required to use the same NSID for a NS client when calling this function each time. This allows the NS client changing its NSID in the lifecycle. For example, the provisioning task may need to switch NSID to provision the keys for different NS clients created afterwards.

```
uint32_t tfm_nsce_save_ctx(uint32_t token)
```

This function should be called when NSPE RTOS schedules out a NS client. The input parameter *token* is returned by *tfm_nsce_acquire_ctx()*. After the context is saved, no secure service call can be made from NSPE until a context is loaded via *tfm_nsce_load_ctx()*.

NSCE integration guide

Enable NSCE in TF-M

To enable NSCE in TF-M, set the build flag *TFM_NS_MANAGE_NSID* to *ON* (default *OFF*).

Support NSCE in an RTOS

For supporting NSCE in an RTOS, the integrator needs to do the following major work:

- Integrate the NSCE API calls into the NS task lifecycle. For example, creating/scheduling/destroying a NS task.
- Manage the assignment for *gid*, *tid* and NSID.

The typical steps are listed below:

- Before programming, the integrator should plan the group assignment for the NS tasks that need to call secure services. If the number of tasks is less than or equal to TF-M non-secure context slots, then different gid can be assigned to each task for taking dedicated context in TF-M. Otherwise, the integrator need to think about grouping the tasks to share the context.
- In the kernel initialization stage, it calls *tfm_nsce_init()* with requested context number to initialize the non-secure context in TF-M. The actual allocated context number will be returned. *0* means initialization failed. The kernel could use different group assignment sets according to the context number allocated to it. TF-M only supports one context for now.
- The kernel calls *tfm_nsce_acquire_ctx()* when creating a new task. This should be done before the new task calls any secure service. A valid token returned should be saved as part of the task context in NSPE RTOS.
- When the kernel schedules in a task with a valid *token* associated, *tfm_nsce_load_ctx()* should be called before resuming the execution of that task. The NSID is specified by the kernel through the *nsid* parameter. The mapping between NSID and task is managed by the kernel. *tfm_nsce_load_ctx()* can be called multiple times without calling *tfm_nsce_save_ctx()* for switching the NSID for the same task (same tid and gid).
- *tfm_nsce_save_ctx()* should be called if the current task has a valid *token* before being switched to another task. Calling *tfm_nsce_load_ctx()* for another task before saving the current context may result in NS context lost in TF-M for the running task.
- When the task is terminated, destroyed or crashed, the kernel should call *tfm_nsce_release_ctx* to make sure the associated resource is back to the pool in TF-M.

Integration example

This is the software module diagram of a typical RTOS/NSCE integration example.

- Built-in Secure Context Manager: An RTOS may have an existing built-in secure context manager with a group of secure context management APIs defined. Let's take RTX which uses Armv8-M TrustZone APIs as the example.

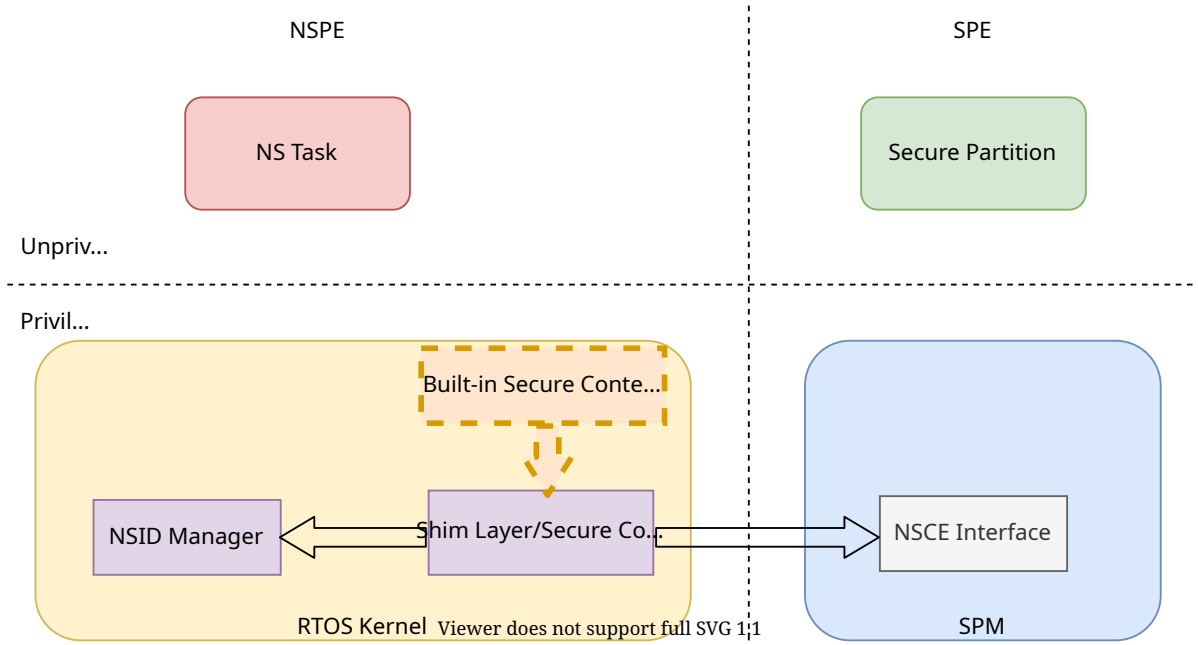


Fig. 3: RTOS/NSCE integration example

Note: RTOS may define the NS task context in the secure side as "secure context". It is the same thing as the "non-secure context" (context for a secure service call from NS client) from TF-M's point of view.

- Shim Layer/Secure Context Manager: If the RTOS has a "Built-in Secure Context Manager", then a shim layer should be provided to translate the built-in API calls into the NSCE API calls. A reference shim layer for RTX TrustZone APIs is here.

  If the RTOS has no existing one, then a "Secure Context Manager" should be implemented based on the NSCE APIs. You can refer to the shim layer example above for the implementation. The timing of calling NSCE APIs in the kernel is introduced in the *Support NSCE in an RTOS* section above.

- NSID Manager: If NSPE RTOS manages the NSID, then this module is used by the secure context manager or shim layer to manage the NSID assignment for the NS tasks. The assignment is implementation defined. A task name based NSID manager is provided as a reference.

Integration notes

- *gid*: It is a *uint8_t* value (valid range is 0 - 255). So, maximum 256 groups (NSCE context slots) are supported by the NSCE interface. TF-M only supports single context for now. So, it is recommended to use single group ID at this stage.
- *tid*: It is a *uint8_t* value (valid range is 0 - 255). Thread ID is used to identify a NS client within a given group. *tid* has no special meaning for TF-M. So, usually the kernel only needs to ensure a NS task has a unique *tid* within a group.
- *gid* and *tid* management: It is the responsibility of NSPE RTOS to manage the assignment of *gid* and *tid*. Based on the explaination above, the *gid* could be assigned as a constant value. And, the *tid* can be increased globally when calling *tfm_nsce_acquire_ctx()* for a new task. Just notice *tid* may overflow.
- NSID management: It is the responsibility of NSPE RTOS to manage the assignment of the NSID for each task. It is highly recommended that a NS client uses the same NSID following

a reboot or update. The binding of a NS client to a specific NSID will ensure the correct access to the assets. For example, the data saved in the protected storage.

- Integrate with the existing secure context management APIs of NSPE RTOS: The NSCE APIs are designed to be compatible with most known existing secure context management APIs. A shim layer is needed to translate the API calls. See the integration example above.
- *tfm_nsce_acquire_ctx()* must be called before calling *tfm_nsce_load_ctx()*, *tfm_nsce_save_ctx()* or *tfm_nsce_release_ctx()*.
- *tfm_nsce_release_ctx()* can be called without calling *tfm_nsce_save_ctx()* ahead.

Copyright (c) 2021, Arm Limited. All rights reserved.

Generic OS migration from Armv7-M to Armv8-M architecture

The purpose of this document is to list a set of requirements needed for migrating a generic OS kernel running on Armv7-M to the Armv8-M architecture.

List of requirements

- If the same OS codebase is used for both Secure and Non Secure builds, it is suggested to put specific code targeted to the Non Secure build under a compile time switch, e.g. `#if (DOMAIN_NS == 1U)`. The OS build system in this case needs to be amended accordingly to support this new switch.
- If the OS implements stack limit checking, the `PSPLIM` register needs to be initialized and properly handled during thread context switch operations.
- If the OS manipulates directly the Link Register, the default Link Register value used in Handler mode transitions needs to be differentiated between Secure and Non Secure builds, i.e. `0xFD` and `0xBC`, respectively.
- If the OS manages the non-secure client identification, please check the *Non-secure Client Extension Integration Guide*.

Copyright (c) 2018-2021, Arm Limited. All rights reserved.

Floating-Point Support

TF-M adds several configuration flags to control Floating point (FP)[1] support in TF-M Secure Processing Environment (SPE) and Non Secure Processing Environment (NSPE).

- Support FP in SPE or NSPE.
- Support FP Application Binary Interface (ABI)[2] types: software, hardware. SPE and NSPE shall use the same FP ABI type.
- Support lazy stacking enable/disable in SPE only, NSPE is not allowed to enable/disable this feature.
- Support GNU Arm Embedded Toolchain[3]. `GNU Arm Embedded Toolchain 10.3- 2021.10` and later version shall be used to mitigate VLLDM instruction security vulnerability[4].
- Support both IPC[5] and SFN[11] models in TF-M.

[1] High-Performance Hardware Support for Floating-Point Operations
[2] Float Point ABI
[3] GNU Arm Embedded Toolchain
[4] VLLDM instruction Security Vulnerability
[5] Arm® Platform Security Architecture Firmware Framework 1.0
[11] FF-M v1.1 Extension

- Support Armv8-M mainline.
- Support isolation level 1,2,3.
- Support Arm Compiler for Embedded[10]. Arm Compiler for Embedded 6.17 and later version shall be used to mitigate VLLDM instruction security vulnerability[Page 1253, 4].
- Does not support use FPU in First-Level Interrupt Handling (FLIH)[6] at current stage.

Please refer to Arm AN521 or AN552 platform as a reference implementation when you enable FP support on your platforms.

Note: Alternatively, if you intend to use FP in your own NSPE application but the TF-M SPE services that you enable do not require FP, you can set the CMake configuration CONFIG_TFM_ENABLE_CP10CP11 to ON and **ignore** any configurations described below.

Note: FPU test issue has not been fixed yet on Musca-S1[7]. When running FPU tests on Musca-S1, secure thread fails to trigger secure interrupt. FPU test is disabled by default on Musca-S1 until the issue is fixed.

Note: GNU Arm Embedded Toolchain 10.3-2021.10 may have issue that reports '-mcpu=cortex-m55' conflicts with '-march=armv8.1-m.main' warning[8]. This issue has been fixed in the later version.

FP ABI type for SPE and NSPE

FP design in Armv8.0-M[9] architecture requires consistent FP ABI types between SPE and NSPE. Furthermore, both sides shall set up CPACR individually when FPU is used. Otherwise, No Coprocessor (NOCP) usage fault will be asserted during FP context switch between security states.

Secure and non-secure libraries are compiled with COMPILER_CP_FLAG and linked with LINKER_CP_OPTION for different FP ABI types. All those libraries shall be built with COMPLIER_CP_FLAG.

If FP ABI types mismatch error is generated during build, pleae check whether the library is compiled with COMPILER_CP_FLAG. Example:

```
target_compile_options(lib
  PRIVATE
    ${COMPILER_CP_FLAG}
)
```

CMake configurations for FP support

The following CMake configurations configure COMPILER_CP_FLAG in TF-M SPE.

- CONFIG_TFM_ENABLE_FP is used to enable/disable FPU usage.

CONFIG_TFM_ENABLE_FP	FP support
off (default)	FP disabled
on	FP enabled

[10] Arm Compiler for Embedded
[6] *Secure Interrupt Integration Guide*
[7] Musca-S1 Test Chip Board
[8] GCC Issue on '-mcpu=cortex-m55' conflicts with '-march=armv8.1-m.main' Warning
[9] Armv8-M Architecture Reference Manual

Note: CONFIG_TFM_FLOAT_ABI depends on CONFIG_TFM_ENABLE_FP. If CONFIG_TFM_ENABLE_FP is set ``CONFIG_TFM_FLOAT_ABI is automatically set to hard.

Note: If you build TF-M SPE with CONFIG_TFM_ENABLE_FP=on and provide your own NSPE application, your own NSPE **must** take care of enabling floating point coprocessors CP10 and CP11 on the NS side to avoid aforementioned NOCP usage fault.

- CONFIG_TFM_LAZY_STACKING is used to enable/disable lazy stacking feature. This feature is only valid for FP hardware ABI type. NSPE is not allowed to enable/disable this feature. Let SPE decide the secure/non-secure shared setting of lazy stacking to avoid the possible side-path brought by flexibility.

CONFIG_TFM_LAZY_STACKING	Description
0FF	Disable lazy stacking
ON (default)	Enable lazy stacking

- CONFIG_TFM_FP_ARCH specifies which FP architecture is available on the target, valid for FP hardware ABI type.

  FP architecture is processor dependent. For GNUARM compiler, example value are: auto, fpv5-d16, fpv5-sp-d16, etc. For armclang, example value are: none, softvfp, fpv5-d16, fpv5-sp-d16, etc.

  This parameter shall be specified by platform in preload.cmake. Please check compiler reference manual and processor hardware manual for more details to set correct FPU configuration for platform.

- CONFIG_TFM_FP_ARCH_ASM specifies the target FPU architecture name shared by Arm Compiler armasm and armlink. It is only used in the --fpu= argument by Arm Compiler and shall be aligned with CONFIG_TFM_FP_ARCH.

  FP architecture is processor dependent. For armasm and armlink, example value are: SoftVFP, FPv5_D16, etc.

  This parameter shall be specified by platform in preload.cmake. Please check compiler reference manual and processor hardware manual for more details to set correct FPU configuration for platform.

Reference

Copyright (c) 2021-2023, Arm Limited. All rights reserved.

Secure Interrupt Integration Guide

Introduction

This document describes how to enable an interrupt in TF-M. The target audiences are mainly platform integrators and Secure Partition developers.

This document assumes that you have read the PSA Firmware Framework (FF-M) v1.0[1] and the FF-M v1.1 extensions[2] thus have knowlege on the terminologies such as Secure Partitions and manifests.

[1] FF-M v1.0 Specification
[2] FF-M v1.1 Extension

Interrupt Handling Model

TF-M supports the two interrupt handling models defined by FF-M:

- First-Level Interrupt Handling (FLIH)

  In this model, the interrupt handling is carried out immediately when the interrupt exception happens.

  The interrupt handling can optionally set an interrupt signal for the Secure Partition Thread to have further data processing.

- Second-Level Interrupt Handling (SLIH)

  In this model, the interrupt handling is deferred after the interrupt exception. The handling occurs in the Secure Partition Thread thus is subject to scheduling.

The FLIH supports handling an interrupt in a bounded time, but very limited APIs are allowed in the FLIH handling because the handling occurs in an special exception context.

The SLIH is deferred and subject to scheduling but all Secure Partition APIs are allowed as the SLIH handling is in the Secure Partition Thread.

Both the FLIH and the SLIH can be used by Secure Partitions which conform to Firmware Framework v1.1.

While the SLIH is the only supported model for Secure Partitions which conform to Firmware Framework v1.0.

Please refer to chapter 6.2 of FF-M v1.1[Page 1255, 1] for more details on the interrupt handling models.

Enabling an Interrupt

To enable an interrupt, you need to do the following:

- Binding the interrupt to a Secure Partition.
- Granting the Secure Partition access permissions to the device of the interrupt.
- Initializing the interrupt.
- Integrating the interrupt handling function

TF-M has two Test Partitions as good examples for both FLIH[3] and SLIH[4]. See also *Enabling the Interrupt Tests* on how to integrate them to platforms.

Binding an Interrupt to a Secure Partition

To bind an interrupt to a Secure Partition, you need to add an item to the `irqs` attribute of the Secure Partition manifest. `irqs` is a list of Interrupt Request (IRQ) assigned to the Secure Partition.

Secure Partitions are not allowed to share IRQs with other Secure Partitions.

Different Firmware Framework versions have different definitions of manifest.

FF-M v1.0

Here is an example manifest of Secure Partitions conform to Firmware Framework version 1.0:

[3] https://git.trustedfirmware.org/TF-M/tf-m-tests.git/tree/tests_reg/test/secure_fw/suites/spm/irq/service/tfm_flih_test_service

[4] https://git.trustedfirmware.org/TF-M/tf-m-tests.git/tree/tests_reg/test/secure_fw/suites/spm/irq/service/tfm_slih_test_service

```
{
  "irqs": [
    {
      "source": 5,
      "signal": "DUAL_TIMER_SIGNAL"
    },
    {
      "source": "TIMER_1_SOURCE",
      "signal": "TIMER_1_SIGNAL"
    }
  ]
}
```

- source

  Required, Unique.

  The source is a string that identifies the interrupt source. It can be a valid exception number or a symbolic name defined in platform codes.

- signal

  Required, Unique.

  The signal attribute is a symbolic name used by TF-M to identify which interrupt is asserted. It is also used by the Secure Partition to receive the interrupt signal by calling psa_wait for interrupt handling.

  It is defined in the Secure Partition header file <psa_manifest/manifestfilename.h> generated by TF-M:

  ```
  #define signal VALUE
  ```

The interrupt handling model is SLIH by default as it is the only supported one for FF-M v1.0.

FF-M v1.1

Here is an example manifest of Secure Partitions conform to Firmware Framework version 1.1:

```
{
  "irqs": [
    {
      "source"  : "TIMER_1_SOURCE",
      "name"    : "TIMER_1",
      "handling": "FLIH"
    },
    {
      "source"  : 5,
      "name"    : "DUAL_TIMER",
      "handling": "SLIH"
    }
  ]
}
```

- source

  The source is the same as the one in Firmware Framework Version 1.0.

- name

  Required, Unique.

  The name is used to construct the following two elements:

  – the interrupt signal symbol: {{name}}_SIGNAL, the equivalent of signal in FF-M v1.0

  - the FLIH Function for handling FLIH IRQs provided by Secure Partition:

    psa_flih_result_t {{name}}_flih(void);

    It is also declared in <psa_manifest/manifestfilename.h>.

- handling

  Required.

  The handling attribute specifies the interrupt handling model and must have one of the following values:

  - FLIH - First-Level Interrupt Handling
  - SLIH - Second-Level Interrupt Handling

Granting Permissions to Devices for Secure Partitions

A secure partition shall be granted two parts of permissions to access a device. One is the Memory Maped I/O (MMIO) region of the device. The other is the driver codes to access the device.

The MMIO Regions

You need to declare the MMIO region in the mmio_regions attributes in the Secure Partition manifest, to enable the Secure Partition to access it.

An MMIO region can be described as either numbered_region or named_region. A numbered region consists of a base address and a size. A named region consists of a string name to describe the region.

Here is an example of named region:

```
{
  "mmio_regions": [
    {
      "name": "TFM_PERIPHERAL_TIMER0",
      "permission": "READ-WRITE"
    }
  ]
}
```

- name

  Required.

  The name attribute is a symbolic name defined by platforms. It is a pointer to structure instance that usually includes the base address and size of the region and some other platform specific attributes that are required to set up permissions to the MMIO region.

  The structure is defined by platforms and the name must be struct platform_data_t.

- permission

  Required.

  The permission attribute must have one of the following values:

  - READ-ONLY
  - READ-WRITE

The Device Drivers

To give permissions of devices drivers to Secure Partitions, it is recommended to put the driver codes to the Partition's CMake library:

```
target_sources(some_partition_lib
    PRIVATE
        some_driver_code.c
)
```

Initializing the Interrupts

Platforms must define an interrupt initialization function for each Secure interrupt.

The prototype of the function is:

```
enum tfm_hal_status_t {source_symbol}_init(void *p_pt,
                                           const struct irq_load_info_t *p_ildi)
```

The {source_symbol} is:

- irq_{source}, if the source attribute of the IRQ in Partition manifest is a number
- Lowercase of source attribute, if source is a symbolic name

For example if the manifest declares "source": 5, then the function name is irq_5_init. If the manifest declares "source" : "TIMER_1_IRQ" then the function name is timer_1_irq_init.

The function will be called by the Framework automatically during initialization. The function can be put in any source file that belongs to SPM, for example a tfm_interrupts.c added to the tfm_spm CMake target.

The initialization of an interrupt must include:

- setting the priority
- ensuring that the interrupt targets the Secure State.
- saving the interrupt information

Setting Priority

The priority of external interrupts must be in the following range: (0, N / 2), where N is the number of configurable priorities. Smaller values have higher priorities.

For example if the number of configurable priority of your interrupt controller is 16, you must use the priorities in range (0, 8) only, boundaries excluded.

Note that these are not the values set into the interrupt controllers. Different platforms may have different values for those priorities. But if you use the NVIC_SetPriority function provided by CMSIS to set priorities, you can pass the values directly.

Platforms have the flexibilities on the assignment of priorities.

Targeting Interrupts to Secure

In single core systems, platform integrators must ensure that the Secure interrupts target to Secure State by setting the Interrupt Controller.

In multi-core systems, this might be optional.

Saving the Interrupt Information

The initialization function is called during Partition loading with the following information:

- p_pt - pointer to Partition runtime struct of the owner Partition
- p_ildi - pointer to irq_load_info_t struct of the interrupt

Platforms must save the information for the future use. See *Integrating the Interrupt Handling Function* for the usage.

The easiest way is to save them in global variables for each interrupt. TF-M provides a struct for saving the information:

```
struct irq_t {
    void                        *p_pt;
    const struct irq_load_info_t *p_ildi;
};
```

Integrating the Interrupt Handling Function

TF-M provides an interrupt handling entry for Secure interrupts:

```
void spm_handle_interrupt(void *p_pt, const struct irq_load_info_t *p_ildi)
```

The p_pt and p_ildi are the information passed to interrupt initialization functions and saved by platforms.

Platforms should call this entry function in the interrupt handlers held in Vector Table with the information saved by the interrupt initialization functions. If the information is saved as global variables, then the interrupt handlers can be put in the same source file that contains the initialization functions.

Here is an example:

```
void TFM_TIMER0_IRQ_Handler(void) /* The handler in Vector Table */
{
    spm_handle_interrupt(p_timer0_pt, p_tfm_timer0_irq_ldinf);
}
```

Enabling the Interrupt Tests

TF-M provides test suites for FLIH and SLIH interrupts respectively. They are disabled by default.

Note: FLIH interrupt test and SLIH interrupt test share the same timer TFM_TIMER0_IRQ thus cannot be enabled at the same time.

To enable the tests, please follow steps in the previous sections. In addition, you need to implement the following APIs of timer control:

- void tfm_plat_test_secure_timer_start(void)
- void tfm_plat_test_secure_timer_clear_intr(void)
- void tfm_plat_test_secure_timer_stop(void)

You shall also select the following flags in platform specific config.cmake to indicate that FLIH and SLIH interrupt tests are supported respectively.

- PLATFORM_FLIH_IRQ_TEST_SUPPORT: platform implements support of FLIH interrupt tests
- PLATFORM_SLIH_IRQ_TEST_SUPPORT: platform implements support of SLIH interrupt tests

The following configurations control SLIH and FLIH interrupt tests:

- TEST_NS_FLIH_IRQ
- TEST_NS_SLIH_IRQ

They can be enabled via build command line or via TEST_NS.

Migrating to Firmware Framework v1.1

Please refer to Migrating Secure Partitions to version 1.1 of FF-M v1.1[Page 1255, 1] .

References

Copyright (c) 2021-2024, Arm Limited. All rights reserved. Copyright (c) 2022 Cypress Semiconductor Corporation (an Infineon company) or an affiliate of Cypress Semiconductor Corporation. All rights reserved.

Platform Provisioning

TF-M stores any data that should be provisioned at the factory in OTP memory. The default is that this OTP memory is actually implemented using on-chip flash, the same that is used to implement the ITS service.

If the lifecycle state is in the TFM_SLC_ASSEMBLY_AND_TEST[1] state (which is the default for non-provisioned boards), then TF-M will attempt to provision the: - HUK instead of booting. It will read the data from the assembly_and_test_prov_data struct, and will then provision it to OTP. The lifecycle state will then transition to TFM_SLC_PSA_ROT_PROVISIONING[1].

If the lifecycle state is in the TFM_SLC_PSA_ROT_PROVISIONING[1] state, then TF-M will attempt to provision the:

- IAK
- boot seed
- implementation id
- certification reference
- bl2 ROTPKs (of which there are up to 4)
- entropy seed

Once all of these have been loaded from the psa_rot_prov_data struct and provisioned to OTP, the LCS will transition to TFM_SLC_SECURED[1]. Note that this provisioning step will be run immediately after the TFM_SLC_ASSEMBLY_AND_TEST[1] provisioning stage if the lifecycle transition completed successfully.

Provisioning development hardware

If TFM_DUMMY_PROVISIONING is enabled in the cmake config (as it is by default), a set of dummy keys / data will be provisioned. The dummy IAK matches the IAK tested by the TF-M tests, and the dummy bl2 ROTPKs match the dummy bl2 keys used by default. TFM_DUMMY_PROVISIONING _MUST_ not be used in production hardware, as the keys are insecure.

Provisioning production hardware

For provisioning of real hardware, firstly TFM_DUMMY_PROVISIONING must be disabled. Then it is required to inject the keys into RAM so they populate the assembly_and_test_prov_data and psa_rot_prov_data structs, at the beginning of the TF-M boot. These structs each require a magic value to be set to be accepted by the provisioning code, which is detailed in platform/ext/common/provisioning.c. Two suggestions for how to do this are:

- Attach a debugger, and inject the values into RAM.

[1] For the definitions of these lifecycle states, please refer to the Platform Security Model https://developer.arm.com/documentation/den0128/0100/

- Flash an image that contains the required data. Care must be taken with this approach that the keys are not left in RAM after provisioning, so a different image (without provisioning data embedded) must be flashed afterwards, without erasing the OTP flash area.

Provisioning on CryptoCell-312 enabled platforms

On boards that have a CC312 accelerator, and that have the default flash-backed OTP disabled by setting PLATFORM_DEFAULT_OTP=OFF in cmake, the CC312 OTP will be used as a backing for the OTP HAL.

Due to the CC312 requiring a power-cycle to transition LCS, you will be prompted to manually power-cycle the board between provisioning stages.

Boards with real OTP memory cannot be reprovisioned - care should be taken that the data being provisioned is the desired data.

Platform-specific OTP backing

If a platform has a medium that is suitable for storing data with OTP semantics (Where a bit cannot transition from a 1 to a 0), such as physical OTP memory, then it can provide a backing for the OTP HAL by implementing the methods described in tfm_plat_otp.h.

Copyright (c) 2020-2022, Arm Limited. All rights reserved.

Adding a new platform

Porting TF-M to a New Hardware

The purpose of this document is to provide a guide on how to integrate TF-M with another hardware platform. This document will give general guidance on how to port a platform on the TF-M build system and which interfaces must exist on the platform for TF-M (S and NS) to run on this new platform.

Prerequisites

Building environnement

Make sure you have a working build environnement and that you can build TF-M on AN521 following the *Build instructions*.

Toolchains and software requirements

Please follow the Getting started guide.

CMSIS Drivers

The TF-M stack requires at least two CMSIS HAL implementations:

- USART
- FLASH

Porting flow

In a nutshell, this should be a 6 iterative steps process:

1. Adding all the mandatory files and expected objects/functions declarations

2. Booting and configuring the core(s)
    - startup(s) code and SystemInit
3. Adding the USART drivers
    - CMSIS HAL
4. Adding the FLASH drivers
    - CMSIS HAL
5. Enabling/Configuring/Disabling features including templated features
    - E.G. NV Counters, attestation, crypto keys....
6. Adding the optional platform SVC handling

    Some platforms may have their own SVC requests in addition to the TF-M built-in ones.
7. Running the regression tests
    - See *Running TF-M on Arm platforms* as an example

File architecture

The platform selection when building TF-M is set via the CMake variable TFM_PLATFORM. This variable holds part of the path to the platform. When using `-DTFM_PLATFORM=arm/mps2/an521` or `-DTFM_PLATFORM=an521` TF-M build system will look for the platform in <TF-M ROOT>/platform/ext/target/arm/mps2/an521. Therefore all hardware dependent code for your platform should go to <TF-M ROOT>/platform/ext/target/.

platform/ext/target

This folder contains a first level of board vendor (such as ARM, STM, NXP, Cypress ....), each folder will usually contain a second level for each board. This second level is not mandatory.

platform/ext/target/<vendor>/[<board name>/]

From now on this will be referred to as the `platform folder`.

platform/ext/common

This folder contains files and folder commons to the platforms, such as the shims to the CMSIS drivers. It also contains the scatter files that can be used for the bl2, tfm_s, tfm_ns partitions.

This folder also contains another folder named template. The latter contains example implementations that are used for platforms by default, but which can be altered or replaced if other functionality is required.

name	description
PLAT-FORM_DEFAULT_ATTEST_H	Use the default implementation of the attestation HAL (default True)
PLAT-FORM_DEFAULT_NV_COUNT	Use the default implementation of the counters in NV (default True)
PLAT-FORM_DEFAULT_CRYPTO_K	Use the default implementation of crypto keys (default True)
PLAT-FORM_DEFAULT_ROTPK	Use the default implementation of the RoT Public Key (default True)
PLATFORM_DEFAULT_IAK	Use the default implementation of the initial attestation key (default True)
PLAT-FORM_DEFAULT_UART_STD	Use the default implementation of the uart for stdout output (default True)
PLAT-FORM_DEFAULT_NV_SEED	Use the default implementation of the NV seed in the RNG (default True)
PLATFORM_DEFAULT_OTP	Use the default implementation of the OTP (default True)

Platform Folder

Description

Depending on the level of integration you want with TF-M some files or information will be mandatory for the build system to build working firmware.

Please note that platform folder provides source for building both *SPE* and *NSPE* parts. The SPE builds directly from the source tree while files necessary for NSPE platform support are installed to <Artifact folder> for building TF-M application as decribed in the *Build instructions*.

Questions to be answered:

- Will the platform use MCUboot as the second stage bootloader?

  BL2/MCUboot provides a secure bootloader that enables simple software upgrades.

  This optional second stage bootloader is set-up via the bl2 target in the CMakelists.txt file (see below).

- Will the platform support the Non-Secure world application?

  A platform can be designed to only support the secure world, in which case we would refer to it as a secure enclave. TF-M build system allows the developer to strip all Non-Secure world related code out of the final image. Most platforms, and especially the ones intended to be generic or to have a Non-Secure application will require Non-Secure world support. In that case a platform shall instruct build system on the file set for exporting to Non-Secure world.

- How does the non-secure world communicate with the secure world?

  TF-M supports running the non-secure world on the same CPU as the secure world, communicating via TrustZone or running the non-secure world on a separate CPU, communicating via a mailbox. The platform is responsible for configuring toolchains with correct CPU and architecture related features for secure and non-secure worlds.

  The architecture for secure world is configured in the cpuarch.cmake file (see below).

- How does the FLASH need to be split between worlds?

  The flash split is very dependent on the support of BL2 and NS world. When porting a new platform, one shall arrange enough flash size for each of them.

  If supporting upgrades (via MCUboot), additional flash area will be required to store the updates before upgrading the whole system.

- How does the RAM need to be split between worlds?

  The RAM split is very dependent on the support of the NS world.

  If you're not porting the platform for a specific project but are enabling the Non-Secure world, you should ensure that you leave enough RAM available for it to run.

Note: TF-M S world size in RAM and Flash varies greatly with different build options.

TF-M project provides metrics of the S world size for existing platforms, which may help to get a rough guide to the sizes needed.

Files

CMakeLists.txt :

(MANDATORY)

This is the entry point for the build system to build your platform on the secure side and also export files to build Non-Secure side.

it must:

- Add a folder to the target platform_region_defs. [*PLATFORM_REGION_DEFS*]

  This folder will contain two files *flash_layout.h* and *region_defs.h*

- Add scatter files to the bl2 and tfm_s targets. [*SCATTER*]

  Please note that TF-M provides a common scatter file for the bl2, tfm_s and tfm_ns targets, which can be used in most cases.

- Add startup files to the bl2 and tfm_s targets. [*STARTUP*]
- Add required sources and includes for the bl2 and tfm_s targets [*SOURCES_INCLUDES*]
- Install all files required for building the platform in the Non-secure application [*INSTALL*]

The installation section expands the common installation script with the platform specific files. The following predefined variables are availble to address the respective subfolders of the target <Artifact folder>.

name	description
INSTALL_INTERFACE_INC_DIR	interface/include - interface header files
INSTALL_INTERFACE_SRC_DIR	interface/src - interface source files
INSTALL_INTERFACE_LIB_DIR	interface/lib - interface libraries
INSTALL_IMAGE_SIGNING_DIR	image_signing tools and files
INSTALL_CMAKE_DIR	CMake modules for Non-secure app build
INSTALL_PLATFORM_NS_DIR	NS platform source files

config.cmake:

(MANDATORY)

This file is used to setup default build configurations for TF-M and platform configurations which have fixed values depending on hardware and software supportness. These configurations should be set as normal CMake variables while others are cache variables.

The platform configurations in the below table are required.

name	description
CONFIG_TFM_USE_TRUSTZONE	Use TrustZone to transition between NSPE and SPE on the same CPU
TFM_MULTI_CORE_TOPOL	NSPE runs on a separate CPU to SPE

The platform configurations in the below table control optional features which rely on platform specific implementation. These features are disabled by default. Platforms shall implement corresponding functionalities and explicitly set the configuration to enable the feature.

name	description
PLATFORM_HAS_ISOLATION_L3_SUPPO	Whether the platform has isolation level 3 support
PLATFORM_HAS_FIRMWARE_UPDATE_S	Wheter the platform has firmware update support
PSA_API_TEST_TARGET	The target platform name of PSA API test
PLATFORM_SVC_HANDLERS	Whether the platform has specific SVC handling

For build configurations, please refer to config_base.cmake.

[*config_cmake*]

cpuarch.cmake:

(MANDATORY)

This file contains hardware information such as the main processor and architecture of the SPE CPU. On single-core platforms, it should be installed to <Artifact folder> for NSPE build. On multi-core platforms, two cpuarch.cmake files should be added.

- a SPE specific cpuarch.cmake used in SPE build
- an NSPE one which should be installed to <Artifact folder> with filename cpuarch.cmake for NSPE build. See *ns/cpuarch_ns.cmake*.

name	description
TFM_SYSTEM_PROCESSOR	The SPE Processor the platform is using
TFM_SYSTEM_ARCHITECTURE	The architecture of the processor
CONFIG_TFM_FP_ARCH	The Float Point architecture flag for toolchain
CONFIG_TFM_FP_ARCH_ASM	The Float Point architecture flag for assembly code

tests/tfm_tests_config.cmake:

(OPTIONAL)

This file contains platform-specific config options for TF-M regression tests. The tests folder should installed to <Artifact folder>/platform for NSPE build. Here are some examples.

name	description
PLATFORM_SLIH_IRQ_TEST_SUPPORT |	Whether the platform has SLIH test support
PLATFORM_FLIH_IRQ_TEST_SUPPORT |	Whether the platform has FLIH test support

tests/psa_arch_tests_config.cmake:

(OPTIONAL)

This file contains platform-specific config options for PSA API tests. Here are some examples.

name	description
PSA_API_TEST_TARGET	The target platform name of PSA API test

startup files:

(MANDATORY)

These files (one for BL2, one for S, one for NS) are the expected startup files. The reset handler should call SystemInit and then should end up calling __START which should be defined as _start if not defined elsewhere.

flash_layout.h:

(MANDATORY)

This file can be anywhere in the platform folder, usually in a sub folder named partition. TF-M doesn't provide a template for this file, common practice is to copy it from another platform (e.g. arm/mps2/an521) and update the following entries.

Note: all size are in bytes

name	description	Requisiteness
FLASH_S_PARTITION_	Size of the Secure partition in flash	Yes
FLASH_NS_PARTITION	Size of the Non-Secure partition in flash	if tfm_ns is built
FLASH_AREA_IMAGE_	Size of the flash sector	if bl2 is built
FLASH_TOTAL_SIZE	Flash total size	Yes
FLASH_BASE_ADDRES	Flash base memory address	if bl2 is built
FLASH_AREA_BL2_OF	BL2 offset in flash	if bl2 is built
FLASH_AREA_BL2_SIZ	BL2 flash size	if bl2 is built
FLASH_PS_AREA_SIZE	Allocated size for the protected storage data in flash	Yes
FLASH_ITS_AREA_SIZ	Allocated size for the internal trusted storage data in flash	Yes
SECURE_IMAGE_OFFSET	Offset of the secure image data in flash	if bl2 is built
FLASH_DEV_NAME	Name as defined in the CMSIS flash drivers	Yes
TFM_HAL_PS_FLASH_	Name as defined in the CMSIS flash drivers	used by protected storage partition
TFM_HAL_PS_SECTOR	Number of physical erase sectors per logical FS block	used by protected storage partition
TFM_HAL_PS_PROGRA	Smallest flash programmable unit in bytes	used by protected storage partition
TFM_HAL_ITS_FLASH_	Name as defined in the CMSIS flash drivers	used by internal trusted storage partition
TFM_HAL_ITS_SECTOI	Number of physical erase sectors per logical ITS block	used by internal trusted storage partition
TFM_HAL_ITS_PROGR	Smallest flash programmable unit in bytes	used by internal trusted storage partition
TFM_NV_COUNTERS_A	Allocated size for the NV counters data in flash	if using TF-M templates

region_defs.h:

(MANDATORY)

This file can be anywhere in the platform folder, usually in a sub folder named partition. TF-M doesn't provide a template for this file, common practice is to copy it from another platform (e.g. arm/mps2/an521) and update the following entries.

General advice: if you don't know beforehand the size you will want for these elements you will have to make it iterative from an abitrary value taken from another platform (e.g. arm/mps2/an521)

Note: all size are in bytes

name	description	Requisiteness
BL2_HEAP_SIZE	Size of the Bootloader (MCUboot) heap	if bl2 is built
BL2_MSP_STACK_SIZE	(if bl2 is built) Size of the Bootloader (MCUboot) Main stack	if bl2 is built
S_HEAP_SIZE	Size of the Secure (S) world Heap	yes
S_MSP_STACK_SIZE	Size of the Secure (S) world Main stack	yes
S_PSP_STACK_SIZE	Size of the Secure (S) world Process stack	no for IPC model
NS_HEAP_SIZE	Size of the Non-Secure (NS) world Heap	if tfm_ns is built
NS_STACK_SIZE	Size of the Non-Secure (NS) world stack	if tfm_ns is built
PSA_INITIAL_ATTEST_M	Size of the buffer that will store the initial attestation	used by initial attestation partition
TFM_ATTEST_BOOT_RE	Size of buffer that can store the encoded list of boot records	used by delegated attestation partition
BL2_HEADER_SIZE	Size of the Header for the Bootloader (MCUboot)	if bl2 is built
BL2_TRAILER_SIZE	Size of the Trailer for the Bootloader (MCUboot)	if bl2 is built
SHARED_SYMBOL_ARE	Size of shared common code between bl2 and tfm_s	if bl2 is built and want to reduce image size

(OPTIONAL)

If the TF-M common linker script is used then:

name	description	Requisiteness
S_CODE_START	Start address for the S code	Yes
S_CODE_SIZE	Size of the S code	Yes
S_DATA_START	Start address for the S data	Yes
S_DATA_SIZE	Size of the S data	Yes
S_RAM_CODE_START	Start address for the S code	if no XIP on flash
S_RAM_CODE_SIZE	Size of the S code	if no XIP on flash

CMSIS_Driver/Config/cmsis_driver_config.h:

(location as defined in CMakeLists.txt)

This file should include the CMSIS drivers implementation headers.

CMSIS_Driver/Config/RTE_Device.h:

(location as defined in CMakeLists.txt)

This is the Run-Time Environnement file from CMSIS, which is there to allow enabling or disabling drivers prior to building. If your platform is designed as a general use platform, this file should contain all the available CMSIS drivers, and you should provide a recommended configuration. If your platform is designed for a specific use-case then you should reference and enable only the mandatory drivers.

CMSIS_Driver/Driver_Flash.c:

(location as defined in CMakeLists.txt)

TF-M relies on CMSIS Drivers, as such it requires the CMSIS functions to be implemented. As a platform owner you can decide to either implement the drivers in the CMSIS functions or to use the CMSIS functions as a shim to your native drivers.

Refer to the CMSIS FLASH documentation.

CMSIS_Driver/Driver_USART.c:

(location as defined in CMakeLists.txt)

TF-M relies on CMSIS Drivers, as such it requires the CMSIS functions to be implemented. As a platform owner you can decide to either implement the drivers in the CMSIS functions or to use the CMSIS functions as a shim to your native drivers.

Refer to the CMSIS USART documentation.

target_cfg.[ch]:

(location as defined in CMakeLists.txt)

It is expected that these files contain all platform specific code related to memory protection (e.g. SAU/PPC/MPC). These functions will not be called by TF-M directly, but are expected to be called from the function tfm_hal_set_up_static_boundaries() in tfm_hal_isolation.c.

tfm_hal_platform.c:

(location as defined in CMakeLists.txt)

Each platform is expected to implement the following API declared in platform/include/tfm_hal_platform.h

```
enum tfm_hal_status_t tfm_hal_platform_init(void);
```

The function will be called before SPM initialization.

tfm_hal_isolation.c:

(location as defined in CMakeLists.txt)

Each platform is expected to implement all the functions declared in platform/include/tfm_hal_isolation.h.

A reference implementation for Armv8-M platforms is provided in platform/ext/common/tfm_hal_isolation_v8m.c. Platforms using the common TF-M linker scripts and scatter files can use it to implement standard TF-M isolation with Armv8-M MPU regions. Platform-specific MPU regions can be appended by defining PLATFORM_STATIC_MPU_REGIONS in the platform's tfm_peripherals_def.h header.

These functions will be called from TF-M.

tfm_platform_system.c:

(location as defined in CMakeLists.txt)

Each platform is expected to implement all the functions declared in platform/include/tfm_platform_system.h.

check_config.cmake:

As a platform owner you may want to enforce some configuration or to prevent the use of unsupported configurations.

This file (CMake format) allows you to do so by allowing you to check for invalid configuration values.

This file is optional.

TF-M build system already provides a generic configuration checker that will be called on top of one provided by the platform owner. The generic checker is located in <TF-M ROOT>/config/.

[*check_config.cmake*]

platform_svc_numbers.h

(OPTIONAL)

If your platform has its own SVC handling, then you need to

- create the platform_svc_numbers.h which defines the platform SVC numbers.

  The bit [7] of the number must be set to 1 to reflect that it is a platform SVC number. The bit [6] indicates whether this SVC should be called from Handler mode or Thread mode. For more details of the bit assignments, please check the svc_num.h. TF-M provides two Macros TFM_SVC_NUM_PLATFORM_THREAD(index) and TFM_SVC_NUM_PLATFORM_HANDLER(index) to easily construct a valid number.

- implement the *platform_svc_handlers* function which handles SVC.
- enable PLATFORM_SVC_HANDLERS config option.

ns/CMakeLists.txt

(MANDATORY)

This is CMake script for building the platform support on NSPE side. It's copied to <Artifact folder> in the installation phase and instructs on how to build **platform_ns** target. The default NSPE build script expects this target definition and extends it with files, common for all TF-M platforms.

Note::
This file shall define and use paths of installed directories in <Artifact folder>, instead of paths in TF-M platform folder.

[*NSCMakeLists.txt*]

ns/cpuarch_ns.cmake

(MANDATORY for multi-core platforms)

This file contains the hardware information for the NSPE CPU. It should be installed to <Artifact folder>/platform for NSPE build, ranamed to cpuarch.cmake.

[*cpuarch.cmake*]

Functions

There are a few functions that need to be declared and properly initialized for TF-M to work. The function declarations can be found in platform/include/tfm_platform_system.h and platform/include/tfm_spm_hal.h.

tfm_platform_hal_system_reset:

This function will in most cases end up calling the NVIC System Reset.

The platform can uninitialize or store some resources before reset.

```
void tfm_platform_hal_system_reset(void);
```

tfm_platform_hal_ioctl:

A single entry point to platform-specific code across the HAL is provided by the IOCTL service.

```
enum tfm_platform_err_t tfm_platform_hal_ioctl(tfm_platform_ioctl_req_t request, psa_invec  *in_vec,␣
↪psa_outvec *out_vec);
```

tfm_hal_get_mem_security_attr:

Required on multi-core platforms only. This function shall fill the security_attr_info_t argument with the current active security configuration.

```
void tfm_hal_get_mem_security_attr(const void *p, size_t s, struct security_attr_info_t *p_attr);
```

tfm_hal_get_secure_access_attr:

Required on multi-core platforms only. This function shall fill the mem_attr_info_t argument with the current active memory configuration of the target S memory region.

```
void tfm_hal_get_secure_access_attr(const void *p, size_t s, struct mem_attr_info_t *p_attr);
```

tfm_hal_get_ns_access_attr:

Required on multi-core platforms only. This function shall fill the mem_attr_info_t argument with the current active memory configuration for the target NS memory region.

```
void tfm_hal_get_ns_access_attr(const void *p, size_t s, struct mem_attr_info_t *p_attr);
```

tfm_hal_irq_clear_pending:

This function clears any pending IRQ.

```
void tfm_hal_irq_clear_pending(uint32_t irq_num);
```

tfm_hal_irq_enable:

This function enable an IRQ.

```
void tfm_hal_irq_enable(uint32_t irq_num);
```

tfm_hal_irq_disable:

This function disable an IRQ.

```
void tfm_hal_irq_disable(uint32_t irq_num);
```

platform_svc_handlers

This function is the platform's SVC handler. It should return the result for callers and the SPM will then return it to the caller.

```
int32_t platform_svc_handlers(uint8_t svc_num, uint32_t *svc_args, uint32_t exc_return);
```

Annex

CMake build system snippets examples

CMakeLists.txt: Defining regions for Secure world platform and all linked to it.

```
target_include_directories(platform_region_defs
    INTERFACE
    <folder name under the platform folder - usually named platform>
)
```

CMakeLists.txt: Scatter files for SPE platform and bootloader

```
target_add_scatter_file(bl2
    $<$<C_COMPILER_ID:ARMClang>:${PLATFORM_DIR}/ext/common/armclang/tfm_common_bl2.
↪sct>
    $<$<C_COMPILER_ID:GNU>:${PLATFORM_DIR}/ext/common/gcc/tfm_common_bl2.ld>
    $<$<C_COMPILER_ID:IAR>:${PLATFORM_DIR}/ext/common/iar/tfm_common_bl2.icf>
)
target_add_scatter_file(tfm_s
    $<$<C_COMPILER_ID:ARMClang>:${PLATFORM_DIR}/ext/common/armclang/tfm_common_s.
↪sct>
    $<$<C_COMPILER_ID:GNU>:${PLATFORM_DIR}/ext/common/gcc/tfm_common_s.ld>
    $<$<C_COMPILER_ID:IAR>:${PLATFORM_DIR}/ext/common/iar/tfm_common_s.icf>
)
```

CMakeLists.txt: Startup files for SPE platform and bootloader

```
target_sources(bl2
    PRIVATE
    ${CMAKE_CURRENT_SOURCE_DIR}/platform/ext/target/<folder to platform>/device/source/
↪startup_<platform name>.c
)
target_sources(tfm_s
    PRIVATE
    ${CMAKE_CURRENT_SOURCE_DIR}/platform/ext/target/<folder to platform>/device/source/
↪startup_<platform name>.c
)
```

CMakeLists.txt: The Secure world platform sources

```
target_include_directories(platform_bl2
    PUBLIC
)
target_include_directories(platform_s
    PUBLIC
)

target_sources(platform_bl2
    PRIVATE
)
target_sources(platform_s
    PRIVATE
)
target_sources(tfm_spm
    PRIVATE
        target_cfg.c
        tfm_hal_isolation.c
        tfm_hal_platform.c
)
```

CMakeLists.txt: installation for the Non-Secure world platform build

```
install(FILES ${PLATFORM_DIR}/ext/common/uart_stdout.c
              native_drivers/arm_uart_drv.c
              native_drivers/timer_cmsdk/timer_cmsdk.c
              cmsis_drivers/Driver_USART.c
              retarget/platform_retarget_dev.c
              cmsis_core/an521_ns_init.c
        DESTINATION ${INSTALL_PLATFORM_NS_DIR})

install(DIRECTORY ${PLATFORM_DIR}/ext/common
                  ${PLATFORM_DIR}/ext/driver
        DESTINATION ${INSTALL_PLATFORM_NS_DIR}/ext)
```

config.cmake

```
set(CONFIG_TFM_USE_TRUSTZONE            ON)
set(TFM_MULTI_CORE_TOPOLOGY             OFF)
set(BL2                                 OFF         CACHE BOOL      "Whether to build BL2")
set(NS                                  FALSE       CACHE BOOL      "Whether to build NS app" FORCE)
```

check_config.cmake

```
function(tfm_invalid_config)
    if (${ARGV})
        string (REPLACE ";" " " ARGV_STRING "${ARGV}")
        string (REPLACE "STREQUAL"      "=" ARGV_STRING "${ARGV_STRING}")
        string (REPLACE "GREATER"       ">" ARGV_STRING "${ARGV_STRING}")
        string (REPLACE "LESS"          "<" ARGV_STRING "${ARGV_STRING}")
        string (REPLACE "VERSION_LESS" "<" ARGV_STRING "${ARGV_STRING}")
        string (REPLACE "EQUAL"         "=" ARGV_STRING "${ARGV_STRING}")
        string (REPLACE "IN_LIST"       "in" ARGV_STRING "${ARGV_STRING}")

        message(FATAL_ERROR "INVALID CONFIG: ${ARGV_STRING}")
    endif()
endfunction()

# Requires armclang >= 6.10.1
tfm_invalid_config((CMAKE_C_COMPILER_ID STREQUAL "ARMClang") AND (CMAKE_C_
↪COMPILER_VERSION VERSION_LESS "6.10.1"))
```

/ns/CMakeLists.txt:

```
add_library(platform_ns)

target_sources(platform_ns
    PRIVATE
        arm_uart_drv.c
        timer_cmsdk.c
        uart_stdout.c
        Driver_USART.c
    PUBLIC
        cmsis_core/startup_an521.c
)

target_include_directories(platform_ns
    PUBLIC
        include
        cmsis
        cmsis_core
)

target_compile_definitions(platform_ns
```

(continues on next page)

(continued from previous page)

```
  PUBLIC
    $<$<BOOL:${PLATFORM_DEFAULT_CRYPTO_KEYS}>:PLATFORM_DEFAULT_CRYPTO_
↪KEYS>
)
```

Copyright (c) 2021-2023, Arm Limited. All rights reserved.

Copyright (c) 2022 Cypress Semiconductor Corporation (an Infineon company) or an affiliate of Cypress Semiconductor Corporation. All rights reserved.

Platform Documentation

This document describes conventions for platform documentation for easy and smooth integration into TF-M online documentation.

Principles

- Minimise TF-M requirements and maximise flexibility to let platform vendors reuse existing documentation.
- Platforms are grouped under vendors' name. There is space to keep general information not specific to any of the vendors' platforms.
- A Table Of Contents (TOC) structures platforms' documentation. There is no defined template for platform documents.

Rules

- Platforms documentation is gathered in */docs/platform* folder. This folder is similar to the platforms implementation folder */platform/ext/target* but does not need to mirror its structure.
- A vendor is represented by a subfolder with *index.rst* as an entry point. This file may contain general materials about a vendor not specific to a particular platform.
- A vendor's *index.rst* shall carry a TOCtree structure with a list of all supported platforms at the top level.
- An explicit platform name is preferred in a vendor's TOCtree instead of use of platform's title by default.

Note: Do not forget to update the summary file after adding a new platform */docs/platform/platform_introduction.rst* included into TF-M introduction.

Example

Assuming a vendor *ABCD123* wants to add and document a new platform *Secure 1024* in TF-M. Here are the basic steps:

1. Create a folder */docs/platform/ABCD123/*
2. Create a platform documentation under */docs/platform/ABCD123*. A natural choice is to put a new platform under */docs/platform/ABCD123/Secure_1024* folder but it's not required.
3. Create a file */docs/platform/ABCD123/index.rst* with a platform entry, presuming it's location in */docs/platform/ABCD123/Secure_1024/index.rst*:

```
################
ABCD123 platforms
################
```

(continues on next page)

(continued from previous page)

```
.. toctree::
Secure 1024 <Secure_1024/index>
```

4. Finally, update */docs/platform/index.rst* with a new vendor name:

```
##############
TF-M Platforms
##############

.. toctree::
   :maxdepth: 2
   :glob:

   Arm <arm/index.rst>
   NXP <nxp/index.rst>
   Cypress <cypress/index.rst>
   ...
   ABCD123 <ABCD123/index.rst>
```

Copyright (c) 2023, Arm Limited. All rights reserved.

Platform deprecation and removal

Process for Platform deprecation and removal

A platform may need to be removed from upstream code due to lack of community interest or it may have reached end of life. The below section calls out the process for removing platform support from TF-M.

1. An email to the TF-M mailing list proposing the removal of the platform.
2. The merit of the proposal will be considered by the maintainers for a period of 4 weeks and community can express their opinion on the same during this time window. The platform owner can veto the proposal and incase of multiple platform owners with differing opinion or community having interest in the platform, then the project maintainer can work with the platform owner and use their discretion to decide on the proposal.
3. Once a decision is made to remove the platform, the platform is considered to be in *deprecated* state as per platform support lifecyle defined here: "https://developer.trustedfirmware.org/w/collaboration/project-maintenance-process/". The platform will be marked as deprecated and the TF-M version after which it will be removed will also be mentioned. Suitable build time or runtime messages needs to be incorporated to the platform to warn about the *deprecation*.
4. The project should strive to keep the *deprecated* platforms building/running till the removal. This relies on platform owners being still actively involved with the project and maintaining the platform.
5. Although this will be the usual process for platform deprecation and eventual removal, the process still leaves room for the platform deprecation to be cancelled after it has been marked as *deprecated* due to evolving project and market requirements. This is left to consensus between project maintainers and platform owner/s.
6. Once a platform has been removed, it can be added back in future and this would follow the same guidelines as for a new platform contribution.

List of Deprecated Platforms

The below list calls out platforms marked for deprecation according to the above process and the platform will be removed soon after the mentioned release.

Deprecated Platform	Removed after release	Comments
arm/mps2/an539	v1.2.0	N.A
arm/mps2/sse-200_aws	v1.3.0	N.A
arm/musca_a	v1.3.0	N.A
arm/mps2/fvp_sse300	v1.4.0	N.A
arm/musca_b1/secure_enclave	v1.6.0	N.A

Copyright (c) 2020-2021, Arm Limited. All rights reserved.

Copyright (c) 2020-2023, Arm Limited. All rights reserved.

Services

Initial Attestation Service Integration Guide

Introduction

TF-M Initial Attestation Service allows the application to prove the device identity during an authentication process to a verification entity. The initial attestation service can create a token on request, which contains a fix set of device specific data.

TF-M Initial Attestation Service by default enables asymmetric key algorithm based attestation (*asymmetric attestation* for short). Symmetric key algorithm based attestation (*symmetric attestation* for short) can be enabled instead by selecting build option SYMMETRIC_INITIAL_ATTESTATION.

- In asymmetric attestation, device must contain an attestation key pair, which is unique per device. The token is signed with the private part of attestation key pair. The public part of the key pair is known by the verification entity. The public key is used to verify the token authenticity.
- In symmetric attestation, device should contain a symmetric attestation key to generate the authentication tag of token content. The verification entity uses the same symmetric key to verify the token authenticity.

The data items in the token used to verify the device integrity and assess its trustworthiness. Attestation key provisioning is out of scope for the attestation service and is expected to take part during manufacturing of the device.

Claims in the initial attestation token

The initial attestation token is formed of claims. A claim is a data item, which is represented in a key - value structure. The following fixed set of claims are included in the token:

- **Auth challenge**: Input object from caller. Can be a single nonce from server or hash of nonce and attested data. It is intended to provide freshness to report and the caller has responsibility to arrange this. Allowed length: 32, 48, 64 bytes. The claim is modeled to be eventually represented by the EAT standard claim nonce. Until such a time as that standard exists, the claim will be represented by a custom claim. Value is encoded as byte string.
- **Instance ID**: It represents the unique identifier of the instance. In the PSA definition it is:
  - a hash of the public attestation key of the instance in asymmetric attestation.

  - hashes of the symmetric attestation key of the instance in symmetric attestation.

  The claim is modeled to be eventually represented by the EAT standard claim UEID of type GUID. Until such a time as that standard exists, the claim will be represented by a custom claim Value is encoded as byte string.

- **Verification service indicator**: Optional, recommended claim. It is used by a Relying Party to locate a validation service for the token. The value is a text string that can be used to locate the service or a URL specifying the address of the service. The claim is modelled to be eventually represented by the EAT standard claim origination. Until such a time as that standard exists, the claim will be represented by a custom claim. Value is encoded as text string.
- **Profile definition**: Optional, recommended claim. It contains the name of a document that describes the 'profile' of the token, being a full description of the claims, their usage, verification and token signing. The document name may include versioning. Custom claim with a value encoded as text string.
- **Implementation ID**: Uniquely identifies the underlying immutable PSA RoT. A verification service can use this claim to locate the details of the verification process. Such details include the implementation's origin and associated certification state. Custom claim with a value encoded as byte string.
- **Client ID**: The partition ID of that secure partition or non-secure thread who called the initial attestation API. Custom claim with a value encoded as a *signed* integer. Negative number represents non-secure caller, positive numbers represents secure callers, zero is invalid.
- **Security lifecycle**: It represents the current lifecycle state of the instance. Custom claim with a value encoded as an integer.
- **Hardware version**: Optional claim. Globally unique number in EAN-13 format identifying the GDSII that went to fabrication, HW and ROM. It can be used to reference the security level of the PSA-ROT via a certification website. Custom claim with a value is encoded as text string.
- **Boot seed**: It represents a random value created at system boot time that will allow differentiation of reports from different system sessions. The size is 32 bytes. Custom claim with a value is encoded as byte string.
- **Software components**: Optional, but required in order to be compliant with the PSA-SM. It represents the software state of the system. The value of the claim is an array of CBOR map entries, with one entry per software component within the device. Each map contains multiple claims that describe evidence about the details of the software component.
- **No software measurements**: Optional, but required if no software component claims are made. In the event that the implementation does not contain any software measurements then it is mandatory to include this claim to indicate this is a deliberate state. Custom claim with a value encoded as an unsigned integer set to 1.

Each software component claim can include the following properties. Any property that is not optional must be included:

- **Measurement type**: Optional claim. It represents the role of the software component. Value is encoded as short(!) text string.
- **Measurement value**: It represents a hash of the invariant software component in memory at start-up time. The value must be a cryptographic hash of 256 bits or stronger. Value is encoded as byte string.
- **Version**: Optional claim. It represents the issued software version. Value is encoded as text string.
- **Signer ID**: Optional claim, but required in order to be compliant with the PSA-SM. It represents the hash of a signing authority public key. Value is encoded as byte string.

- **Measurement description**: Optional claim. It represents the way in which the measurement value of the software component is computed. Value is encoded as text string containing an abbreviated description (name) of the measurement method.

Initial attestation token (IAT) data encoding

The initial attestation token is planned to be aligned with future version of Entity Attestation Token format. The token is encoded according to the CBOR format and signed according to COSE standard.

Code structure

The PSA interface for the Initial Attestation Service is located in interface/include. The only header to be included by applications that want to use functions from the PSA API is psa/initial_attestation.h.

The TF-M Initial Attestation Service source files are located in secure_fw/partitions/initial_attestation.

Service source files

- **CBOR library**
  - lib/ext/qcbor This library is used to create a proper CBOR token. It can be used on 32-bit and 64-bit machines. It was designed to suite constrained devices with low memory usage and without dynamic memory allocation. Its source code is fetched automatically during the build configuration step from an external repository: QCBOR library.
  - <qcbor_src>/inc/qcbor/qcbor_encode.h: Public API documentation of CBOR library (encoding).
  - <qcbor_src>/inc/qcbor/qcbor_decode.h: Public API documentation of CBOR library (decoding).
- **COSE library:**
  - lib/ext/t_cose: This library is used to sign a CBOR token and create the COSE header and signature around the initial attestation token. Only a subset of the COSE standard is implemented. The COSE_Sign1 and COSE_Mac0 (only available in TF-M fork) signature schemas are supported.
  - It is a fork of this external t_cose library.
  - lib/ext/t_cose/src/t_cose_crypto.h: Expose an API to bind t_cose library with available crypto library in the device.
  - lib/ext/t_cose/crypto_adapters/t_cose_psa_crypto.c: Implements the exposed API and ports t_cose to the PSA Crypto API.
- **Initial Attestation Service:**
  - attest_core.c : Implements core functionalities such as implementation of APIs, retrieval of claims and token creation.
  - attest_token_encode.c: Implements the token creation functions such as start and finish token creation and adding claims to the token.
  - attest_asymmetric_key.c: Calculate the Instance ID value based on asymmetric initial attestation key.
  - tfm_attest.c: Implements the SPM abstraction layer, and bind the attestation service to the SPM implementation in TF-M project.
  - tfm_attest_req_mngr.c: Includes the initialization entry of attestation service and handles attestation service requests in IPC model.

  - attest_symmetric_key.c: Calculate the Instance ID value based on symmetric initial attestation key.

Service interface definitions

- **Boot loader interface**: The attestation service might include data in the token about the distinct software components in the device. This data is provided by the boot loader and must be encoded in the TLV format, definition is described below in the boot loader interface paragraph. Possible claims in the boot status are describe above in the software components paragraph.
- **Hardware abstraction layer:**
  - Headers are located in platform/include folder.
  - tfm_attest_hal.h: Expose an API to get the following claims: security lifecycle, verification service indicator, profile definition.
  - tfm_plat_boot_seed.h: Expose an API to get the boot seed claim.
  - tfm_plat_device_id.h: Expose an API to get the following claims: implementation ID, hardware version.
- **SPM interface:**
  - attestation.h: Expose an API to bind attestation service to an SPM implementation.
- **PSA interface:**
  - psa/initial_attestation.h: Public API definition of initial attestation service.
- **Crypto interface:**
  - t_cose_crypto.h: Expose an API to bind the t_cose implementation to any cryptographic library.

PSA interface

The TF-M Initial Attestation Service exposes the following PSA interface:

```
psa_status_t
psa_initial_attest_get_token(const uint8_t *auth_challenge,
                             size_t         challenge_size,
                             uint8_t        *token_buf,
                             size_t         token_buf_size,
                             size_t         *token_size);

psa_status_t
psa_initial_attest_get_token_size(size_t challenge_size,
                                  size_t *token_size);
```

The caller must allocate a large enough buffer, where the token is going to be created by Initial Attestation Service. The size of the created token is highly dependent on the number of software components in the system and the provided attributes of these. The psa_initial_attest_get_token_size() function can be called to get the exact size of the created token.

System integrators might need to port these interfaces to a custom secure partition manager implementation (SPM). Implementations in TF-M project can be found here:

- interface/src/tfm_attest_api.c: interface implementation.

Secure Partition Manager (SPM) interface

The Initial Attestation Service defines the following interface towards the secure partition manager (SPM). System integrators **must** port this interface according to their SPM implementation.

```
enum psa_attest_err_t
attest_get_boot_data(uint8_t major_type, void *ptr, uint32_t len);

enum psa_attest_err_t
attest_get_caller_client_id(int32_t *caller_id);
```

- attest_get_boot_data(): Service can retrieve the relevant data from shared memory area between boot loader and runtime software. It might be the case that only SPM has direct access to the shared memory area, therefore this function can be used to copy the service related data from shared memory to a local memory buffer. In TF-M implementation this function must be called during service initialization phase, because the shared memory region is deliberately overlapping with secure main stack to spare some memory and reuse this area during execution. If boot loader is not available in the system to provide attributes of software components then this function must be implemented in a way that just initialize service's memory buffer to:

```
struct shared_data_tlv_header *tlv_header = (struct shared_data_tlv_header *)ptr;
tlv_header->tlv_magic   = 2016;
tlv_header->tlv_tot_len = sizeof(struct shared_data_tlv_header *tlv_header);
```

- attest_get_caller_client_id(): Retrieves the ID of the caller thread.
- tfm_client.h: Service relies on the following external definitions, which must be present or included in this header file:

```
typedef struct psa_invec {
    const void *base;
    size_t len;
} psa_invec;

typedef struct psa_outvec {
    void *base;
    size_t len;
} psa_outvec;
```

Hardware abstraction layer

The following API definitions are intended to retrieve the platform specific claims. System integrators **must** implement these interface according to their SoC and software design. Detailed definition of the claims are above in the claims in the initial attestation token paragraph.

- tfm_attest_hal_get_security_lifecycle(): Get the security lifecycle of the device.
- tfm_attest_hal_get_verification_service(): Get the verification service indicator for initial attestation.
- tfm_attest_hal_get_profile_definition(): Get the name of the profile definition document for initial attestation.
- tfm_plat_get_boot_seed(): Get the boot seed, which is a constant random number during a boot cycle.
- tfm_plat_get_implementation_id: Get the implementation ID of the device.
- tfm_plat_get_cert_ref: Get the hardware version of the device.

Boot loader interface

It is **recommended** to have a secure boot loader in the boot chain, which is capable of measuring the runtime firmware components (calculates the hash value of firmware images) and provide other attributes of these (version, type, etc). If the used boot loader is not capable of sharing these information with the runtime software then the BOOT_DATA_AVAILABLE compiler flag **must** be set to OFF (see *Related compile time options*).

The shared data between boot loader and runtime software is TLV encoded. The definition of TLV structure is described in bl2/include/tfm_boot_status.h. The shared data is stored in a well known location in secure internal memory and this is a contract between boot loader and runtime SW.

The structure of shared data must be the following:

- At the beginning there must be a header: struct shared_data_tlv_header This contains a magic number and a size field which covers the entire size of the shared data area including this header.

```
struct shared_data_tlv_header {
    uint16_t tlv_magic;
    uint16_t tlv_tot_len;
};
```

- The header is followed by the entries which are composed from an entry header structure: struct shared_data_tlv_entry and the data. In the entry header there is a type and a length field. The tlv_type field identifies the consumer of the entry in the runtime software and specify the subtype of that data item. The tlv_len field covers the length of the data (not including the size of the entry header).

  After the entry header structure comes the actual data.

```
struct shared_data_tlv_entry {
    uint16_t tlv_type;
    uint16_t tlv_len;
};
```

- Arbitrary number and size of data entry can be in the shared memory area.

The figure below gives of overview about the tlv_type field in the entry header. The tlv_type always composed from a major and minorbnumber. Major number identifies the addressee in runtime software, which the databentry is sent to. Minor number used to encode more info about the data entry. The actual definition of minor number could change per major number. In case of boot status data, which is going to be processed by initial attestation service the minor number is split further to two part: sw_module and claim. The sw_module identifies the SW component in the system which the data item belongs to and the claim part identifies the exact type of the data.

tlv_type description:

```
|-------------------------------------------- |
|                tlv_type (16 bits)           |
|---------------------------------------------|
|  tlv_major(4 bits)  |  tlv_minor(12 bits)   |
|---------------------------------------------|
| MAJOR_IAS  | sw_module(6 bits) | claim(6 bits) |
|---------------------------------------------|
| MAJOR_CORE |         TBD                     |
|---------------------------------------------|
```

Overall structure of shared data:

```
-------------------------------------------------------------
| Magic number(uint16_t) | Shared data total length(uint16_t) |
-------------------------------------------------------------
| Major_type(4 bits) | Minor_type(12 bits) | Length(uint16_t) |
-------------------------------------------------------------
|                       Raw data                            |
-------------------------------------------------------------
|                          .                                |
|                          .                                |
|                          .                                |
```

(continues on next page)

(continued from previous page)

```
-------------------------------------------------------------
| Major_type(4 bits) | Minor_type(12 bits) | Length(uint16_t) |
-------------------------------------------------------------
|                         Raw data                          |
-------------------------------------------------------------
```

Crypto interface

Asymmetric key algorithm based attestation

Device **must** contain an asymmetric key pair. The private part of it is used to sign the initial attestation token. Current implementation supports only the ECDSA P256 signature over SHA256. The public part of the key pair is used to create the key identifier (kid) in the unprotected part of the COSE header. The kid is used by verification entity to look up the corresponding public key to verify the signature in the token. The *t_cose* part of the initial attestation service implements the signature generation and kid creation. But the actual calculation of token's hash and signature is done by the Crypto service in the device. System integrators might need to re-implement the following functions if they want to use initial attestation service with a different cryptographic library than Crypto service:

- t_cose_crypto_pub_key_sign(): Calculates the signature over a hash value.
- t_cose_crypto_get_ec_pub_key(): Get the public key to create the key identifier.
- t_cose_crypto_hash_start(): Start a multipart hash operation.
- t_cose_crypto_hash_update(): Add a message fragment to a multipart hash operation.
- t_cose_crypto_hash_finish():Finish the calculation of the hash of a message.

Interface needed by verification code:

- t_cose_crypto_pub_key_verify(): Verify the signature over a hash value.

Key handling

The provisioning of the initial attestation key is out of scope of the service and this document. It is assumed that device maker provisions the unique asymmetric key pair during the manufacturing process. Software integrators **must** make sure that TFM_BUILTIN_KEY_SLOT_IAK is available via the Crypto service, which will then be used by the Attestation partition to perform the required signing operations via the PSA crypto interface.

Symmetric key algorithm based attestation

Device **must** contain a symmetric key to generate the authentication tag of the initial attestation token. A key identifier (kid) can be encoded in the unprotected part of the COSE header. It helps verification entity look up the symmetric key to verify the authentication tag in the token.

The *t_cose* part of the initial attestation service implements the authentication tag generation. The authentication tag generation is done by the Crypto service. System integrators might need to re-implement the following functions if platforms provide a different cryptographic library than Crypto service:

- t_cose_crypto_hmac_sign_setup(): Set up a multi-part HMAC calculation operation.
- t_cose_crypto_hmac_update(): Add a message fragment to a multi-part HMAC operation.
- t_cose_crypto_hmac_sign_finish(): Finish the calculation of the HMAC of a message.

Interface needed by verification code:

- t_cose_crypto_hmac_verify_setup(): Set up a multi-part HMAC verification operation.
- t_cose_crypto_hmac_verify_finish(): Finish the verification of the HMAC of a message.

It also requires the same hash operations as listed in asymmetric key algorithm based initial attestation above, in attestation test cases.

Key handling

The provisioning of the initial attestation key is out of scope of the service and this document. It is assumed that device maker provisions the symmetric key during the manufacturing process. The following API is defined to retrieve the symmetric attestation key from platform layer. Software integrators **must** port this interface according to their SoC design and make sure that key is available by Crypto service:

- tfm_plat_get_symmetric_iak(): Get the symmetric initial attestation key raw data.
- tfm_plat_get_symmetric_iak_id(): Get the key identifier of the symmetric initial attestation key. The key identifier can be used as kid parameter in COSE header. Optional.

Note: Asymmetric initial attestation and symmetric initial attestation may share the same HAL APIs in future development.

Initial Attestation Service compile time options

There is a defined set of flags that can be used to compile in/out certain service features. The CommonConfig.cmake file sets the default values of those flags. The list of flags are:

- ATTEST_INCLUDE_OPTIONAL_CLAIMS: Include also the optional claims to the attestation token. Default value: ON in base configure and profile large while OFF in profile small and medium.
- ATTEST_INCLUDE_COSE_KEY_ID: COSE key-id is an optional field in the COSE unprotected header. Key-id is calculated and added to the COSE header based on the value of this flag. Default value: OFF.
- ATTEST_CLAIM_VALUE_CHECK: Check attestation claims against hard-coded values found in platform/ext/common/template/attest_hal.c. Default value is OFF. Set to ON in a platform's CMake file if the attest HAL is not yet properly ported to it.
- SYMMETRIC_INITIAL_ATTESTATION: Select symmetric initial attestation. Default value: OFF.
- ATTEST_INCLUDE_TEST_CODE: The initial attestation implementation is instrumented with additional test code. This is required in order to run some of the initial attestation regression tests. These tests are not required to be run by platform integrators, and are only meant to be used for development or modification of the initial attestation implementation. Enabling this option enables T_COSE_DISABLE_SHORT_CIRCUIT_SIGN which will short circuit the signing operation. Default value: OFF.
- ATTEST_STACK_SIZE- Defines the stack size of the Initial Attestation Partition. This value mainly depends on the build type(debug, release and minisizerel) and compiler.

Related compile time options

- BOOT_DATA_AVAILABLE: The boot data is expected to be present in the shared data area between the boot loader and the runtime firmware when it's ON. Otherwise, when it's OFF does not check the content of the shared data area but instead assumes that the TLV header is present and valid (the magic number is correct) and there are no data entries. Its default value depends on the BL2 flag.

Comparison of asymmetric and symmetric algorithm based token authentication

The symmetric key based authentication requires a more complex infrastructure for key management. Symmetric keys must be kept secret because they are sensitive asset, like the private

key in case of asymmetric cryptographic algorithms. The main difference is that private keys are only stored on device, with proper hardware protection against external access, but symmetric keys must be known by both party (device and verifier), so they must also be stored in a central server of a relying party (who verifies the tokens). If keys are revealed then devices can be impersonated. If the database with the symmetric keys becomes compromised then all corresponding devices become untrusted. Since a centralised database of symmetric keys may need to be network connected, this can be considered to be a valuable target for attackers. The advantage of ECDSA based token authentication is that sensitive assets is only stored one place (in the device) and only one unique key per device. So if a device is compromised then only that single device become untrusted. In this case, the database of the relying party contains the corresponding public keys, which are not considered to be a confidential assets, so they can be shared with anybody. This shows the main advantage of asymmetric based authentication, because verification of attestation tokens can be done by a third party, such as cloud service providers (CSP). Thus Device Maker (DM) or Chip Maker (CM) does not need to operate such a service.

	Symmetric	Asymmetric
Authentication mode	HMAC over SHA256	ECDSA P256 over SHA256
Crypto key type in HW	Symmetric key	ECDSA private key (secp256r1)
Secrets are stored	Device and database	Device only
Verification database contains	Same symmetric key	Public keys
COSE authentication tag in the token	COSE_Mac0	COSE_Sign1
Verification entity	CM or DM, who provisioned the symmetric key	Can be anybody: third party provisioning service, cloud service provider, CM, DM

Verification

Regression test

The initial attestation token is verified by the attestation test suite in `test/secure_fw/suites/attestation`. The test suite is responsible for verifying the token signature and parsing the token to verify its encoding and the presence of the mandatory claims. This test suite can be executed on the device. It is part of the regression test suite. The test suite is configurable in the `test/secure_fw/suites/attestation/attest_token_test_values.h` header file. In this file there are two attributes for each claim which are configurable (more details in the header file):

- Requirements of presence: optional or mandatory
- Expected value: Value check can be disabled or expected value can be provided here.

For initial attestation tests, the built-in IAK is used. Initial attestation regression test verifies the IAT generated by initial attestation service with the exported public key.

iat-verifier

There is another possibility to verify the attestation token. This addresses the off-device testing when the token is already retrieved from the device and verification is done on the requester side. There is a Python script for this purpose in the tf-m-tools repo called iat-verifier. It does the same checking as the attestation test suite. The following steps describe how to simulate an off-device token verification on a host computer. It is described how to retrieve an initial attestation token when TF-M code is executed on FVP and how to use the iat-verifier script to check the token. This example assumes that user has license for DS-5 and FVP models:

- Build TF-M with any of the ConfigRegression*.cmake build configurations for MPS2 AN521 platform. More info in *tfm_build_instruction*.
- Lunch FVP model in DS-5. More info in *Run TF-M examples on Arm platforms*.
- Set a breakpoint in test/secure_fw/suites/attestation/attest_token_test.c in decode_test_internal(..) after the token_main_alt(..) returned, i.e. on line 859. Execute the code in the model until the breakpoint hits second time. At this point the console prints the test case name:
  - For asymmetric initial attestation, the console prints ECDSA signature test of attest token.
  - For symmetric initial attestation, the console prints Symmetric key algorithm based Initial Attestation test.
- At this point the token resides in the model memory and can be dumped to host computer.
- The ADDRESS and SIZE attributes of the initial attestation token is stored in the completed_token local variable. Their value can be extracted in the (x)=Variables debug window.
- Apply commands below in the Commands debug window to dump the token in binary format to the host computer:
  - For asymmetric initial attestation dump memory <PATH>/iat_01.cbor <ADDRESS> +<SIZE>
  - For symmetric initial attestation dump memory <PATH>/iat_hmac_02.cbor <ADDRESS> +<SIZE>
- Execute commands below on the host computer to verify the token:
  - For asymmetric initial attestation check_iat -p -K -k platform/ext/common/template/tfm_initial_attestation_key.pem <PATH>/iat_01.cbor
  - For symmetric initial attestation check_iat -m mac -p -K -k platform/ext/common/template/tfm_symmetric_iak.key <PATH>/iat_hmac_02.cbor
- Documentation of the iat-verifier can be found in the tf-m-tools-iat-verifer.

Copyright (c) 2018-2022, Arm Limited. All rights reserved.

Crypto Service Integration Guide

Introduction

The TF-M Crypto service allows Non Secure world applications and Secure services to use cryptographic functionalities such as symmetric ciphering, message signing and verification, asymmetric encryption and decryption, cryptographic hashes, message authentication codes (MACs), key derivation and agreement, authenticated encryption with associated data (AEAD). It exposes the PSA Crypto APIs[1].

The secure service resides in the Crypto partition as a single entry point and is made of different components:

- An interface layer that exposes the PSA Crypto API to either NS or S entities and is implemented in interface/src/tfm_crypto_api.c. The interface is based on the Uniform Secure Service Signature and communicates with the Secure Partition Manager available in TF-M
- An init module secure_fw/partitions/crypto/crypto_init.c that implements functionalities requested by TF-M during the initialisation phase, and an API dispatcher that at runtime receives the requests from the interface and dispatches them to the component that processes that particular API request

[1] PSA Crypto APIs: https://armmbed.github.io/mbed-crypto/html/

- A set of components that process cryptographic API requests, each component dispatching to a subset of functionalities, i.e. AEAD, Asymmetric, Ciphering, Hashing, Key derivation, Key management, MACs, and Random Number Generation
- An alloc module secure_fw/partitions/crypto/crypto_alloc.c that manages the partition secure memory, storing multipart application contexts, input / outputs of the APIs being requested, inaccessible from NS or other secure partitions
- A library abstraction module secure_fw/partitions/crypto/crypto_library.c which is used to abstract the details of the cryptographic library used by the service *backend* to provide the actual implementation of the crypto functionalities. The backend library must expose the PSA Crypto APIs, and must provide support to encode *key ownership* into key identifiers. This is not standardized by the PSA Crypto APIs so it must be provided as an extension to the APIs. The backend library also needs to provide the subsystem for key storage and retrieval, and, in case, the interface to the cryptographic accelerator of the underlying platform, using the PSA cryptoprocessor driver interface specification[2]. For this reason, it must provide a mechanism to access platform *builtin* keys, and permanent key slots using the *TF-M Internal Trusted Storage (ITS) service*, if available.

Code structure

The PSA interfaces for the Crypto service are located in interface/include. The only header to be included by applications that want to use functions from the PSA API is psa/crypto.h. The TF-M Crypto service source files are located in secure_fw/partitions/crypto.

PSA interfaces

The TF-M Crypto service exposes the PSA interfaces detailed in the header psa/crypto.h. This header, in turn, includes several other headers which are not meant to be included directly by user applications. For a detailed description of the PSA API interface, please refer to the comments in the psa/crypto.h header itself.

Service source files

A brief description of what is implemented by each source file is as below:

- crypto_cipher.c : Dispatcher for symmetric crypto operations
- crypto_hash.c : Dispatcher for hash operations
- crypto_mac.c : Dispatcher for MAC operations
- crypto_aead.c : dispatcher for AEAD operations
- crypto_key_derivation.c : Dispatcher for key derivation and key agreement operations
- crypto_key_management.c : Dispatcher for key management operations towards the key slot management system provided by the backend library
- crypto_rng.c : Dispatcher for the random number generation requests
- crypto_asymmetric.c : Dispatcher for message signature/verification and encryption/decryption using asymmetric crypto
- crypto_init.c : Init module for the service. The modules stores also the internal buffer used to allocate temporarily the IOVECs needed, which is not required in case of SFN model. The size of this buffer is controlled by the CRYPTO_IOVEC_BUFFER_SIZE config define
- crypto_library.c : Library abstractions to interface the dispatchers towards the underlying library providing *backend* crypto functions. Currently this only supports the Mbed TLS library. In particular, the mbed TLS library requires to provide a static buffer to

[2] PSA cryptoprocessor driver interface: https://github.com/Mbed-TLS/mbedtls/blob/development/docs/proposed/psa-driver-interface.md

be used as heap for its internal allocation. The size of this buffer is controlled by the CRYPTO_ENGINE_BUF_SIZE config define

- crypto_alloc.c : Takes care of storing multipart operation contexts in a secure memory not visible outside of the crypto service. The CRYPTO_CONC_OPER_NUM config define determines how many concurrent contexts are supported at once. In a multipart operation, the client view of the contexts is much simpler (i.e. just an handle), and the Alloc module keeps track of the association between handles and contexts
- tfm_crypto_api.c : This module is contained in interface/src and implements the PSA Crypto API client interface exposed to both S/NS clients. This module allows a configuration option CONFIG_TFM_CRYPTO_API_RENAME to be set to 1 in case the NS environment or integrators want to rename the API symbols exported by the TF-M Crypto service. The renaming adds a default prefix, tfm_crypto__ to all functions. The prefix can be changed editing the interface file. This config option is for the NS environment or integration setup only, hence it is not accessible through the TF-M config
- tfm_mbedcrypto_alt.c : This module is specific to the Mbed TLS[3] library integration and provides some alternative implementation of Mbed TLS APIs that can be used when a optimised profile is chosen. Through the _ALT mechanism it is possible to replace at link time default implementations available in Mbed TLS with the ones available in this file

Note: The _ALT mechanism will be deprecated in future releases of the Mbed TLS library

Considerations on service configuration

Crypto *backend* library configuration

The TF-M Crypto service relies on a cryptographic library to provide the functionalities specific by the PSA Crypto API spec and the PSA cryptoprocessor driver interface spec. At the moment, the only supported library is mbed TLSPage 1288, 3.

The configuration of the backend library is supplied using the TFM_MBEDCRYPTO_CONFIG_PATH and TFM_MBEDCRYPTO_PSA_CRYPTO_CONFIG_PATH config option that point to configuration headers following the legacy Mbed TLS configuration scheme or the new PSA based configuration scheme.

Platforms can specify an extra config file by setting the TFM_MBEDCRYPTO_PLATFORM_EXTRA_CONFIG_F variable (which is a wrapper around the MBEDTLS_USER_CONFIG_FILE option). This is preferred for platform configuration over TFM_MBEDCRYPTO_CONFIG_PATH and TFM_MBEDCRYPTO_PSA_CRYPTO_CONFIG_PATH as it does not interfere with config changes due to TFM Profile.

Note: The default entropy source configured for Mbed TLS is MBEDTLS_ENTROPY_NV_SEED with a unique seed. For production devices, an alternative hardware entropy source can be specified using the config option MBEDTLS_ENTROPY_HARDWARE_ALT

Note: Starting from Mbed TLS 3.3.0, the Python package jsonschema must be available when building as it is required by the autogen framework for the driver integrations into the PSA Crypto core and driver wrapper modules

Crypto service build time options

[3] Mbed TLS library: https://www.trustedfirmware.org/projects/mbed-tls/

- CRYPTO_STACK_SIZE : Defines the stack size of the Crypto Secure Partition. This value might depend on several parameters such as the build type, the compiler being used, the cryptographic functionalities that are enabled at build time
- CRYPTO_<COMPONENT>_MODULE_ENABLED : A series of defines, one per each <COMPONENT> that processes cryptographic operations, that are used to disable modules at build time. Each define corresponds to a component as described in *the components list*.

Crypto service *builtin* keys integration

A detailed description of how the service interacts with *builtin* keys is available in the tfm_builtin_key_loader *design document*.

Note: The crypto service integration with builtin keys relies on implementation details of Mbed TLS that are not standardized in the spec and might change between releases due to ongoing work[4]

References

Copyright (c) 2018-2023, Arm Limited. All rights reserved.

TF-M Internal Trusted Storage Service Integration Guide

Introduction

TF-M Internal Trusted Storage (ITS) service implements PSA Internal Trusted Storage APIs.

The service is backed by hardware isolation of the flash access domain and relies on hardware to isolate the flash area from access by the Non-secure Processing Environment, as well as the Application Root of Trust at higher levels of isolation.

The current ITS service design relies on hardware abstraction provided by TF-M. The ITS service provides a non-hierarchical storage model, as a filesystem, where all the assets are managed by a linearly indexed list of metadata.

The design addresses the following high level requirements as well:

- **Confidentiality** - Resistance to unauthorised accesses through hardware/software attacks. Assumed to be provided by the internal flash device, backed by hardware isolation.
- **Access Authentication** - Mechanism to establish requester's identity (a non-secure entity, secure entity, or a remote server).
- **Integrity** - Resistance to tampering by attackers with physical access is assumed to be provided by the internal flash device itself, while resistance to tampering by Non-secure or App RoT attackers also requires hardware isolation.
- **Reliability** - Resistance to power failure scenarios and incomplete write cycles.
- **Configurability** - High level of configurability to scale up/down memory footprint to cater for a variety of devices with varying requirements.
- **Performance** - Optimized to be used for resource constrained devices with very small silicon footprint, the PPA (power, performance, area) should be optimal.

Current ITS Service Limitations

[4] Interface for platform keys: https://github.com/ARM-software/psa-crypto-api/issues/550

- **Fragmentation** - The current design does not support fragmentation, as an asset is stored in a contiguous space in a block. This means that the maximum asset size can only be up-to a block size. Each block can potentially store multiple assets. A delete operation implicitly moves all the assets towards the top of the block to avoid fragmentation within block. However, this may also result in unutilized space at the end of each block.
- **Non-hierarchical storage model** - The current design uses a non-hierarchical storage model, as a filesystem, where all the assets are managed by a linearly indexed list of metadata. This model locates the metadata in blocks which are always stored in the same flash location. That increases the number of writes in a specific flash location as every change in the storage area requires a metadata update.
- **Protection against physical storage medium failure** - Complete handling of inherent failures of storage mediums (e.g. bad blocks in a NAND based device) is not supported by the current design.
- **Lifecycle management** - Currently, it does not support any subscription based keys and certificates required in a secure lifecycle management. Hence, an asset's validity timestamp can not be invalidated based on the system time.
- **Provisioning vs user/device data** - In the current design, all assets are treated in the same manner. In an alternative design, it may be required to create separate partitions for provisioning content and user/device generated content. This is to allow safe update of provisioning data during firmware updates without the need to wipe out the user/device generated data.

Code Structure

TF-M Internal Trusted Storage service code is located in secure_fw/partitions/internal_trusted_storage/ and is divided as follows:

- Core files
- Flash filesystem interfaces
- Flash interfaces

The PSA ITS interfaces for the TF-M ITS service are located in interface/include/psa.

PSA Internal Trusted Storage Interfaces

The TF-M ITS service exposes the following mandatory PSA ITS interfaces version 1.0:

```
psa_status_t psa_its_set(psa_storage_uid_t uid, size_t data_length, const void *p_data, psa_storage_
↪create_flags_t create_flags);
psa_status_t psa_its_get(psa_storage_uid_t uid, size_t data_offset, size_t data_size, void *p_data, size_
↪t *p_data_length);
psa_status_t psa_its_get_info(psa_storage_uid_t uid, struct psa_storage_info_t *p_info);
psa_status_t psa_its_remove(psa_storage_uid_t uid);
```

These PSA ITS interfaces and TF-M ITS types are defined and documented in interface/include/psa/storage_common.h, interface/include/psa/internal_trusted_storage.h, and interface/include/tfm_its_defs.h

Core Files

- tfm_its_req_mngr.c - Contains the ITS request manager implementation which handles all requests which arrive to the service. This layer extracts the arguments from the input and output vectors, and it calls the internal trusted storage layer with the provided parameters.
- tfm_internal_trusted_storage.c - Contains the TF-M internal trusted storage API implementations which are the entry points to the ITS service. Constructs a filesystem configuration

using information from the ITS HAL, allocates a filesystem context for ITS and makes appropriate FS calls. Also handles requests from the PS partition with a separate FS config and context.

- its_utils.c - Contains common and basic functionalities used across the ITS service code.

Flash Filesystem Interface

- flash_fs/its_flash_fs.h - Abstracts the flash filesystem operations used by the internal trusted storage service. The purpose of this abstraction is to have the ability to plug-in other filesystems or filesystem proxies (supplicant).
- flash_fs/its_flash_fs.c - Contains the its_flash_fs implementation for the required interfaces.
- flash_fs/its_flash_fs_mblock.c - Contains the metadata block manipulation functions required to implement the its_flash_fs interfaces in flash_fs/its_flash_fs.c.
- flash_fs/its_flash_fs_dblock.c - Contains the data block manipulation functions required to implement the its_flash_fs interfaces in flash_fs/its_flash_fs.c.

The system integrator **may** replace this implementation with its own flash filesystem implementation or filesystem proxy (supplicant).

Flash Interface

The ITS filesystem flash interface is defined by struct its_flash_fs_ops_t in flash_fs/its_flash_fs.h.

Implementations of the ITS filesystem flash interface for different types of storage can be found in the `internal_trusted_storage/flash directory.

- flash/its_flash.h - Helper header that includes the correct ITS flash interface implementation for the target, and abstracts the allocation of different flash device types.
- flash/its_flash_nand.c - Implements the ITS flash interface for a NAND flash device, on top of the CMSIS flash interface implemented by the target. This implementation writes entire block updates in one-shot, so the CMSIS flash implementation **must** be able to detect incomplete writes and return an error the next time the block is read.
- flash/its_flash_nor.c - Implements the ITS flash interface for a NOR flash device, on top of the CMSIS flash interface implemented by the target.
- flash/its_flash_ram.c - Implements the ITS flash interface for an emulated flash device using RAM, on top of the CMSIS flash interface implemented by the target.

The CMSIS flash interface **must** be implemented for each target based on its flash controller.

The ITS flash interface depends on target-specific definitions from platform/ext/target/<TARGET_NAME>/partition/flash_layout.h. Please see the *Internal Trusted Storage Service HAL* section for details.

ITS Service Integration Guide

This section describes mandatory (i.e. **must** implement) or optional (i.e. **may** implement) interfaces which the system integrator has to take in to account in order to integrate the internal trusted storage service in a new platform.

Flash Interface

For ITS service operations, a contiguous set of blocks must be earmarked for the internal trusted storage area. The design requires either 2 blocks, or any number of blocks greater than or equal to 4. Total number of blocks can not be 0, 1 or 3. This is a design choice limitation to provide power failure safe update operations.

Maximum Asset Size

An asset is stored in a contiguous space in a logical filesystem block. The maximum size of an asset can be up-to the size of the data block. Typically, each logical block corresponds to one physical flash erase sector (the smallest unit that can erased), but the TFM_HAL_ITS_SECTORS_PER_BLOCK configuration below allows a number of contiguous erase sectors to form one logical block.

Internal Trusted Storage Service HAL

The ITS service requires the platform to implement the ITS HAL, defined in platform/include/tfm_hal_its.h.

The following C definitions in the HAL are mandatory, and must be defined by the platform in a header named flash_layout.h:

- TFM_HAL_ITS_FLASH_DRIVER - Defines the identifier of the CMSIS Flash ARM_DRIVER_FLASH object to use for ITS. It must have been allocated by the platform and will be declared extern in the HAL header.
- TFM_HAL_ITS_PROGRAM_UNIT - Defines the size of the ITS flash device's physical program unit (the smallest unit of data that can be individually programmed to flash). It must be equal to TFM_HAL_ITS_FLASH_DRIVER.GetInfo()->program_unit, but made available at compile time so that filesystem structures can be statically sized. Valid values are powers of two between 1 and the flash sector size, inclusive.

The following C definitions in the HAL may optionally be defined by the platform in the flash_layout.h header:

- TFM_HAL_ITS_FLASH_AREA_ADDR - Defines the base address of the dedicated flash area for ITS.
- TFM_HAL_ITS_FLASH_AREA_SIZE - Defines the size of the dedicated flash area for ITS in bytes.
- TFM_HAL_ITS_SECTORS_PER_BLOCK - Defines the number of contiguous physical flash erase sectors that form a logical erase block in the filesystem. The typical value is 1, but it may be increased so that the maximum required asset size will fit in one logical block.

If any of the above definitions are not provided by the platform, then the tfm_hal_its_fs_info() HAL API must be implemented instead. This function is documented in tfm_hal_its.h.

The sectors reserved to be used for Internal Trusted Storage **must** be contiguous.

Internal Trusted Storage Service Optional Platform Definitions

The following optional platform definitions may be defined in flash_layout.h:

- ITS_RAM_FS_SIZE - Defines the size of the RAM FS buffer when using the RAM FS emulated flash implementation. The buffer must be at least as large as the area earmarked for the filesystem by the HAL.
- ITS_FLASH_NAND_BUF_SIZE - Defines the size of the write buffer when using the NAND flash implementation. The buffer must be at least as large as a logical filesystem block.
- ITS_MAX_BLOCK_DATA_COPY - Defines the buffer size used when copying data between blocks, in bytes. If not provided, defaults to 256. Increasing this value will increase the memory footprint of the service.

More information about the flash_layout.h content, not ITS related, is available in *Details for the platform/ext folder* along with other platform information.

ITS Service Build Definitions

The ITS service uses a set of C definitions to compile in/out certain features, as well as to configure certain service parameters. When using the TF-M build system, these definitions are controlled

by build flags of the same name. The config/config_base.cmake file sets the default values of those flags, but they can be overwritten based on platform capabilities by setting them in platform/ext/target/<TARGET_NAME>/config.cmake. The list of ITS service build definitions is:

- ITS_CREATE_FLASH_LAYOUT- this flag indicates that it is required to create an ITS flash layout. If this flag is set, ITS service will generate an empty and valid ITS flash layout to store assets. It will erase all data located in the assigned ITS memory area before generating the ITS layout. This flag is required to be set if the ITS memory area is located in a non-persistent memory. This flag can be set if the ITS memory area is located in a persistent memory without a valid ITS flash layout in it. That is the case when it is the first time in the device life that the ITS service is executed.
- ITS_VALIDATE_METADATA_FROM_FLASH- this flag allows to enable/disable the validation mechanism to check the metadata store in flash every time the flash data is read from flash. This validation is required if the flash is not hardware protected against data corruption.
- ITS_RAM_FS- setting this flag to ON enables the use of RAM instead of the persistent storage device to store the FS in the Internal Trusted Storage service. This flag is OFF by default. The ITS regression tests write/erase storage multiple time, so enabling this flag can increase the life of flash memory when testing. If this flag is set to ON, ITS_RAM_FS_SIZE must also be provided. This specifies the size of the block of RAM to be used to simulate the flash.

  Note: If this flag is disabled when running the regression tests, then it is recommended that the persistent storage area is erased before running the tests to ensure that all tests can run to completion. The type of persistent storage area is platform specific (eFlash, MRAM, etc.) and it is described in corresponding flash_layout.h

  ---

- ITS_MAX_ASSET_SIZE - Defines the maximum asset size to be stored in the ITS area. This size is used to define the temporary buffers used by ITS to read/write the asset content from/to flash. The memory used by the temporary buffers is allocated statically as ITS does not use dynamic memory allocation.
- ITS_NUM_ASSETS - Defines the maximum number of assets to be stored in the ITS area. This number is used to dimension statically the filesystem metadata tables in RAM (fast access) and flash (persistent storage). The memory used by the filesystem metadata tables is allocated statically as ITS does not use dynamic memory allocation.
- ITS_BUF_SIZE- Defines the size of the partition's internal data transfer buffer. If not provided, then ITS_MAX_ASSET_SIZE is used to allow asset data to be copied between the client and the filesystem in one iteration. Reducing the buffer size will decrease the RAM usage of the partition at the expense of latency, as data will be copied in multiple iterations. *Note:* when data is copied in multiple iterations, the atomicity property of the filesystem is lost in the case of an asynchronous power failure.
- ITS_STACK_SIZE- Defines the stack size of the Internal Trusted Storage Secure Partition. This value mainly depends on the platform specific flash drivers, the build type (Debug, Release and MinSizeRel) and compiler.

Copyright (c) 2019-2022, Arm Limited. All rights reserved. Copyright (c) 2020, Cypress Semiconductor Corporation. All rights reserved.

Platform Service Integration Guide

Introduction

TF-M Platform service is a trusted service which allows secure partitions and non-secure applications to interact with some platform-specific components. There are a number of features which requires some interaction with platform-specific components which are at the same time

essential for the security of the system. Therefore, those components need to be handled by a secure partition which is part of the trusted compute base.

These platform-specific components include system reset, power management, Debug, GPIO, etc.

TF-M Platform interfaces

The TF-M interfaces for the Platform service are located in interface/include/. The TF-M Platform service source files are located in secure_fw/partitions/platform.

TF-M Platform service

The Platform service interfaces and types are defined and documented in interface/include/tfm_platform_api.h

- platform_sp.h/c : These files define and implement functionalities related to the platform service
- tfm_platform_api.c : This file implements tfm_platform_api.h functions to be called from the secure partitions. This is the entry point when the secure partitions request an action to the Platform service (e.g system reset).

Platform HAL

The Platform Service relies on a platform-specific implementation to perform some functionalities. Mandatory functionality (e.g. system reset) that are required to be implemented for a platform to be supported by TF-M have their dedicated HAL API functions. Additional platform-specific services can be provided using the IOCTL function call.

For API specification, please check: platform/include/tfm_platform_system.h

An implementation is provided in all the supported platforms. Please, check platform/ext/target/<SPECIFIC_TARGET_FOLDER>/services/src/tfm_platform_system.c for examples.

The API **must** be implemented by the system integrators for their targets.

IOCTL

A single entry point to platform-specific code across the HAL is provided by the IOCTL service and HAL function:

```
enum tfm_platform_err_t tfm_platform_hal_ioctl(tfm_platform_ioctl_req_t request,
                                               psa_invec *in_vec,
                                               psa_outvec *out_vec);
```

A request type is provided by the client, with additional parameters contained in the optional in_vec parameter. An optional output buffer can be passed to the service in out_vec. An IOCTL request type not supported on a particular platform should return TFM_PLATFORM_ERR_NOT_SUPPORTED

Non-Volatile counters

The Platform Service provides an abstracted service for exposing the NV counters to secure partitions or non-secure callers. The following operations are supported:

- Increment a counter.
- Read a counter value to a preallocated buffer.

```
enum tfm_platform_err_t
tfm_platform_nv_counter_increment(uint32_t counter_id);

enum tfm_platform_err_t
```

(continues on next page)

(continued from previous page)

```
tfm_platform_nv_counter_read(uint32_t counter_id,
                             uint32_t size, uint8_t *val);
```

The range of counters id is defined in : platform/include/tfm_plat_nv_counters.h

For Level 2,3 isolation implementations, secure partitions in the Application Root of Trust, should have TFM_PLATFORM_SERVICE set as a dependency for access to the NV counter API.

Current Service Limitations

- **system reset** - The system reset functionality is only supported in isolation level 1. Currently the mechanism by which PSA-RoT services should run in privileged mode in level 3 is not in place due to an ongoing work in TF-M Core. So, the NVIC_SystemReset call performed by the service is expected to generate a memory fault when it tries to access the SCB->AIRCR register in level 3 isolation.

Copyright (c) 2018-2022, Arm Limited. All rights reserved.

Protected Storage Service Integration Guide

Introduction

TF-M Protected Storage (PS) service implements PSA Protected Storage APIs.

The service is usually backed by hardware isolation of the flash access domain and, in the current version, relies on hardware to isolate the flash area from non-secure access. In absence of hardware isolation, the secrecy and integrity of data is still maintained.

The PS service implements an AES-GCM based AEAD encryption policy, as a reference, to protect data integrity and authenticity.

The PS reuses the non-hierarchical filesystem provided by the TF-M Internal Trusted Storage service to store encrypted, authenticated objects.

The design addresses the following high level requirements as well:

- **Confidentiality** - Resistance to unauthorised accesses through hardware/software attacks.
- **Access Authentication** - Mechanism to establish requester's identity (a non-secure entity, secure entity, or a remote server).
- **Integrity** - Resistant to tampering by either the normal users of a product, package, or system or others with physical access to it. If the content of the protected storage is changed maliciously, the service is able to detect it.
- **Reliability** - Resistant to power failure scenarios and incomplete write cycles.
- **Configurability** - High level configurability to scale up/down memory footprint to cater for a variety of devices with varying security requirements.
- **Performance** - Optimized to be used for resource constrained devices with very small silicon footprint, the PPA (power, performance, area) should be optimal.

Current PS Service Limitations

- **Asset size limitation** - An asset is stored in a contiguous space in a block/sector. Hence, the maximum asset size can be up-to the size of the data block/sector. Detailed information about the maximum asset size can be found in the section *Maximum asset size* below.

- **Fragmentation** - The current design does not support fragmentation, as an asset is stored in a contiguous space in a block. Each block can potentially store multiple assets. A delete operation implicitly moves all the assets towards the top of the block to avoid fragmentation within block. However, this may also result in unutilized space at the end of each block.
- **Non-hierarchical storage model** - The current design uses a non-hierarchical storage model, as a filesystem, where all the assets are managed by a linearly indexed list of metadata. This model locates the metadata in blocks which are always stored in the same flash location. That increases the number of writes in a specific flash location as every change in the storage area requires a metadata update.
- **PSA internal trusted storage API** - In the current design, the service does not use the PSA Internal Trusted Storage API to write the rollback protection values stored in the internal storage.
- **Protection against physical storage medium failure** - Complete handling of inherent failures of storage mediums (e.g. bad blocks in a NAND based device) is not supported by the current design.
- **Key diversification** - In a more robust design, each asset would be encrypted through a different key.
- **Lifecycle management** - Currently, it does not support any subscription based keys and certificates required in a secure lifecycle management. Hence, an asset's validity timestamp can not be invalidated based on the system time.
- **Provisioning vs user/device data** - In the current design, all assets are treated in the same manner. In an alternative design, it may be required to create separate partitions for provisioning content and user/device generated content. This is to allow safe update of provisioning data during firmware updates without the need to wipe out the user/device generated data.

Code Structure

Protected storage service code is located in secure_fw/partitions/protected_storage/ and is divided as follows:

- Core files
- Cryptographic interfaces
- Non-volatile (NV) counters interfaces

The PSA PS interfaces for PS service are located in interface/include/psa

PSA Protected Storage Interfaces

The PS service exposes the following mandatory PSA PS interfaces, version 1.0:

```
psa_status_t psa_ps_set(psa_storage_uid_t uid, size_t data_length, const void *p_data, psa_storage_
↪create_flags_t create_flags);
psa_status_t psa_ps_get(psa_storage_uid_t uid, size_t data_offset, size_t data_size, void *p_data, size_
↪t *p_data_length);
psa_status_t psa_ps_get_info(psa_storage_uid_t uid, struct psa_storage_info_t *p_info);
psa_status_t psa_ps_remove(psa_storage_uid_t uid);
uint32_t psa_ps_get_support(void);
```

For the moment, it does not support the extended version of those APIs.

These PSA PS interfaces and PS TF-M types are defined and documented in interface/include/psa/protected_storage.h, interface/include/psa/storage_common.h and interface/include/tfm_ps_defs.h

Core Files

- tfm_ps_req_mngr.c - Contains the PS request manager implementation which handles all requests which arrive to the service. This layer extracts the arguments from the input and output vectors, and it calls the protected storage layer with the provided parameters.
- tfm_protected_storage.c - Contains the TF-M protected storage API implementations which are the entry points to the PS service.
- ps_object_system.c - Contains the object system implementation to manage all objects in PS area.
- ps_object_table.c - Contains the object system table implementation which complements the object system to manage all object in the PS area. The object table has an entry for each object stored in the object system and keeps track of its version and owner.
- ps_encrypted_object.c - Contains an implementation to manipulate encrypted objects in the PS object system.
- ps_utils.c - Contains common and basic functionalities used across the PS service code.
- ps_filesystem_interface.c - Contains the interface to directly use ITS implementation of services.

Flash Filesystem and Flash Interfaces

The non-hierarchical filesystem and flash interfaces reside under Internal Trusted Storage service directory. The way PS service uses them depends on the config option TFM_PARTITION_INTERNAL_TRUSTED_STORAGE.

When TF-M Internal trusted storage service is active (i.e. TFM_PARTITION_INTERNAL_TRUSTED_STORAGE = ON), all file system and storage interfaces are built as part of ITS service library. Thus, the PS service stores encrypted, authenticated objects by making service calls to the ITS service.

When TF-M Internal trusted storage service is disabled (i.e. TFM_PARTITION_INTERNAL_TRUSTED_STORAGE = OFF), all file system and storage interfaces are built as part of PS service library. Thus, the PS service stores encrypted, authenticated objects by making standard function calls to the file system within its own partition code.

The ITS filesystem and flash interfaces and their implementation can be found in secure_fw/partitions/internal_trusted_storage/flash_fs and secure_fw/partitions/internal_trusted_storage/flash respectively. More information about the filesystem and flash interfaces can be found in the *ITS integration guide*.

The ITS service implementation in secure_fw/partitions/internal_trusted_storage/tfm_internal_trusted_storage.c, constructs a filesystem configuration for Protected Storage based on target-specific definitions from the Protected Storage HAL. Please see the *Protected Storage Service HAL* section for details of these.

Cryptographic Interface

- crypto/ps_crypto_interface.h - Abstracts the cryptographic operations for the protected storage service.
- crypto/ps_crypto_interface.c - Implements the PS service cryptographic operations with calls to the TF-M Crypto service.

Non-volatile (NV) Counters Interface

The current PS service provides rollback protection based on NV counters. PS defines and implements the following NV counters functionalities:

- nv_counters/ps_nv_counters.h - Abstracts PS non-volatile counters operations. This API detaches the use of NV counters from the TF-M NV counters implementation, provided by the platform, and provides a mechanism to compile in a different API implementation for test

purposes. A PS test suite **may** provide its own custom implementation to be able to test different PS service use cases.

- nv_counters/ps_nv_counters.c - Implements the PS NV counters interfaces based on TF-M NV counters implementation provided by the platform.

PS Service Integration Guide

This section describes mandatory (i.e. **must** implement) or optional (i.e. **may** implement) interfaces which the system integrator have to take in to account in order to integrate the protected storage service in a new platform.

Maximum Asset Size

An asset is stored in a contiguous space in a block/sector. The maximum size of an asset can be up-to the size of the data block/sector minus the object header size (PS_OBJECT_HEADER_SIZE) which is defined in ps_object_defs.h. The PS_OBJECT_HEADER_SIZE changes based on the PS_ENCRYPTION flag status.

Protected Storage Service HAL

The PS service requires the platform to implement the PS HAL, defined in platform/include/tfm_hal_ps.h.

The following C definitions in the HAL are mandatory, and must be defined by the platform in a header named flash_layout.h:

- TFM_HAL_PS_FLASH_DRIVER - Defines the identifier of the CMSIS Flash ARM_DRIVER_FLASH object to use for PS. It must have been allocated by the platform and will be declared extern in the HAL header.
- TFM_HAL_PS_PROGRAM_UNIT - Defines the size of the PS flash device's physical program unit (the smallest unit of data that can be individually programmed to flash). It must be equal to TFM_HAL_PS_FLASH_DRIVER.GetInfo()->program_unit, but made available at compile time so that filesystem structures can be statically sized. Valid values are powers of two between 1 and the flash sector size, inclusive.

The following C definitions in the HAL may optionally be defined by the platform in the flash_layout.h header:

- TFM_HAL_PS_FLASH_AREA_ADDR - Defines the base address of the dedicated flash area for PS.
- TFM_HAL_PS_FLASH_AREA_SIZE - Defines the size of the dedicated flash area for PS in bytes.
- TFM_HAL_PS_SECTORS_PER_BLOCK - Defines the number of contiguous physical flash erase sectors that form a logical erase block in the filesystem. The typical value is 1, but it may be increased so that the maximum required asset size will fit in one logical block.

If any of the above definitions are not provided by the platform, then the tfm_hal_ps_fs_info() HAL API must be implemented instead. This function is documented in tfm_hal_ps.h.

The sectors reserved to be used for Protected Storage **must** be contiguous sectors starting at TFM_HAL_PS_FLASH_AREA_ADDR.

The design requires either 2 blocks, or any number of blocks greater than or equal to 4. Total number of blocks can not be 0, 1 or 3. This is a design choice limitation to provide power failure safe update operations.

Protected Storage Service Optional Platform Definitions

The following optional platform definitions may be defined in flash_layout.h:

- PS_RAM_FS_SIZE - Defines the size of the RAM FS buffer when using the RAM FS emulated flash implementation. The buffer must be at least as large as the area earmarked for the filesystem by the HAL.
- PS_FLASH_NAND_BUF_SIZE - Defines the size of the write buffer when using the NAND flash implementation. The buffer must be at least as large as a logical filesystem block.

More information about the flash_layout.h content, not ITS related, is available in *Details for the platform/ext folder* along with other platform information.

TF-M NV Counter Interface

To have a platform independent way to access the NV counters, TF-M defines a platform NV counter interface. For API specification, please check: platform/include/tfm_plat_nv_counters.h

The system integrators **may** implement this interface based on the target capabilities and set the PS_ROLLBACK_PROTECTION flag to compile in the rollback protection code.

> **Note:** If this flag is enabled, the lifecycle of the PS service depends on the shorter write endurance of the assets storage device and the NV counters storage device.

Secret Platform Unique Key

The encryption policy relies on a secret hardware unique key (HUK) per device. It is system integrator's responsibility to provide an implementation which **must** be a non-mutable target implementation. For API specification, please check: platform/include/tfm_plat_crypto_keys.h

A stub implementation is provided in platform/ext/common/template/crypto_keys.c

Non-Secure Identity Manager

TF-M core tracks the current client IDs running in the secure or non-secure processing environment. It provides a dedicated API to retrieve the client ID which performs the service request.

Non-secure Client Extension Integration Guide provides further details on how client identification works.

PS service uses that TF-M core API to retrieve the client ID and associate it as the owner of an asset. Only the owner can read, write or delete that asset based on the creation flags.

The *integration guide* provides further details of non-secure implementation requirements for TF-M.

Cryptographic Interface

The reference encryption policy is built on AES-GCM, and it **may** be replaced by a vendor specific implementation.

The PS service abstracts all the cryptographic requirements and specifies the required cryptographic interface in secure_fw/partitions/protected_storage/crypto/ps_crypto_interface.h

The PS service cryptographic operations are implemented in secure_fw/partitions/protected_storage/crypto/ps_crypto_interface.c, using calls to the TF-M Crypto service.

PS Service Build Definitions

The PS service uses a set of C definitions to compile in/out certain features, as well as to configure certain service parameters. When using the TF-M build system, these definitions are controlled by build flags of the same name. The config/config_base.cmake file sets the default values of those flags, but they can be overwritten based on platform capabilities by setting them in platform/ext/target/<TARGET_NAME>/config.cmake. The list of PS service build definitions is:

- PS_ENCRYPTION- this flag allows to enable/disable encryption option to encrypt the protected storage data.
- PS_CRYPTO_AEAD_ALG - this flag indicates the AEAD algorithm to use for authenticated encryption in Protected Storage.

Note: For GCM/CCM it is essential that IV doesn't get repeated. If this flag is set to PSA_ALG_GCM or PSA_ALG_CCM, PS_ROLLBACK_PROTECTION must be enabled to protect against IV rollback.

- PS_CREATE_FLASH_LAYOUT- this flag indicates that it is required to create a PS flash layout. If this flag is set, PS service will generate an empty and valid PS flash layout to store assets. It will erase all data located in the assigned PS memory area before generating the PS layout. This flag is required to be set if the PS memory area is located in a non-persistent memory. This flag can be set if the PS memory area is located in a persistent memory without a valid PS flash layout in it. That is the case when it is the first time in the device life that the PS service is executed.
- PS_VALIDATE_METADATA_FROM_FLASH- this flag allows to enable/disable the validation mechanism to check the metadata store in flash every time the flash data is read from flash. This validation is required if the flash is not hardware protected against malicious writes. In case the flash is protected against malicious writes (i.e embedded flash, etc), this validation can be disabled in order to reduce the validation overhead.
- PS_ROLLBACK_PROTECTION- this flag allows to enable/disable rollback protection in protected storage service. This flag takes effect only if the target has non-volatile counters and PS_ENCRYPTION flag is on.
- PS_RAM_FS- setting this flag to ON enables the use of RAM instead of the persistent storage device to store the FS in the Protected Storage service. This flag is OFF by default. The PS regression tests write/erase storage multiple time, so enabling this flag can increase the life of flash memory when testing. If this flag is set to ON, PS_RAM_FS_SIZE must also be provided. This specifies the size of the block of RAM to be used to simulate the flash.

Note: If this flag is disabled when running the regression tests, then it is recommended that the persistent storage area is erased before running the tests to ensure that all tests can run to completion. The type of persistent storage area is platform specific (eFlash, MRAM, etc.) and it is described in corresponding flash_layout.h

---

- PS_MAX_ASSET_SIZE - Defines the maximum asset size to be stored in the PS area. This size is used to define the temporary buffers used by PS to read/write the asset content from/to flash. The memory used by the temporary buffers is allocated statically as PS does not use dynamic memory allocation.
- PS_NUM_ASSETS - Defines the maximum number of assets to be stored in the PS area. This number is used to dimension statically the object table size in RAM (fast access) and flash (persistent storage). The memory used by the object table is allocated statically as PS does not use dynamic memory allocation.
- PS_TEST_NV_COUNTERS- this flag enables the virtual implementation of the PS NV counters interface in test/secure_fw/suites/ps/secure/nv_counters of the tf-m-tests repo, which emulates NV counters in RAM, and disables the hardware implementation of NV counters provided by the secure service. This flag is enabled by default, but has no effect when the secure regression test is disabled. This flag can be overridden to OFF when building the regression tests. In this case, the PS rollback protection test suite will not be built, as it relies on extra functionality provided by the virtual NV counters to simulate different rollback scenarios. The remainder of the PS test suites will run using the hardware NV counters. Please note that running the tests in this configuration will quickly increase the

hardware NV counter values, which cannot be decreased again. Overriding this flag from its default value of OFF when not building the regression tests is not currently supported.

- PS_STACK_SIZE- Defines the stack size of the Protected Storage Secure Partition. This value mainly depends on the build type(debug, release and minisizerel) and compiler.

Copyright (c) 2018-2024, Arm Limited. All rights reserved. Copyright (c) 2020, Cypress Semiconductor Corporation. All rights reserved.

Adding Secure Partition

Terms and abbreviations

This document uses the following terms and abbreviations.

Table 36: term table

Term	Meaning
FF-M	Firmware Framework for M
ID	Identifier
IPC	Interprocess communication
IPC model	The secure IPC framework
irqs	Interrupt requests
MMIO	Memory Mapped I/O
PSA	Platform Security Architecture
RoT	Root of Trust
SFN	Secure Function
SFN model	Secure Function model
SID	RoT Service ID
SP	Secure Partition
SPM	Secure Partition Manager
TF-M	Trusted firmware M

Introduction

Secure Partition is an execution environment that provides the following functions to Root of Trust (RoT) Services:

- Access to resources, protection of its own code and data.
- Mechanisms to interact with other components in the system.

Each Secure Partition is a single thread of execution and the smallest unit of isolation.

This document mainly describes how to add a secure partition in TF-M and focuses on the configuration, manifest, implement rules. The actual source-level implementation is not included in this document.

Note: If not otherwise specified, the steps are identical for IPC and SFN model.

The IPC and SFN model conforms to the *PSA Firmware Framework for M (FF-M) v 1.1* changes. Refer to PSA Firmware Framework specification and Firmware Framework for M 1.1 Extensions for details.

Process

The main steps to add a secure partition are as follows:

- *Add source folder*
- *Add manifest*
- *Update the Build System*
- *Implement the RoT services*

Add source folder

Add a source folder under <TF-M base folder>/secure_fw/partitions for the new secure partition (Let's take example as the folder name):

This folder should include those parts:

- Manifest file
- CMake configuration files
- Source code files

Add manifest

Each Secure Partition must have resource requirements declared in a manifest file. The Secure Partition Manager (SPM) uses the manifest file to assemble and allocate resources within the SPE. The manifest includes the following:

- A Secure Partition name.
- A list of implemented RoT Services.
- Access to other RoT Services.
- Memory requirements.
- Scheduling hints.
- Peripheral memory-mapped I/O regions and interrupts.

Note: The current manifest format in TF-M is "yaml" which is different from the requirement of PSA FF.

Note: The users can use LOW, NORMAL and HIGH to determine the priority of the Secure Partition in manifest. They are replaced by 01, 02 and 03 auotmatically when parsing manifest lists for section naming.

Here is a manifest reference example for the IPC model:

Note: To use SFN model, the user needs to replace "model": "IPC" to "model": "SFN". The user also needs to remove the attribute "entry_point", and optionally replace it with "entry_init".

```
{
  "psa_framework_version": 1.1,
  "name": "TFM_SP_EXAMPLE",
  "type": "APPLICATION-ROT",
  "priority": "NORMAL",
  "model": "IPC",
  "entry_point": "tfm_example_main",
  "stack_size": "0x0200",
  "services" : [
    {
```

(continues on next page)

(continued from previous page)

```
    "name": "ROT_A",
    "sid": "0x000000E0",
    "non_secure_clients": true,
    "connection_based": true,
    "version": 1,
    "version_policy": "STRICT"
    "mm_iovec": "disable"
  }
 ],
 "mmio_regions": [
  {
   "name": "TFM_PERIPHERAL_A",
   "permission": "READ-WRITE"
  }
 ],
 "irqs": [
  {
   "source": "TFM_A_IRQ",
   "name": "A_IRQ",
   "handling": "SLIH"
  }
 ]
 "dependencies": [
  "TFM_CRYPTO",
  "TFM_INTERNAL_TRUSTED_STORAGE_SERVICE"
 ]
}
```

Update manifest list

The <TF-M base folder>/tools/tfm_manifest_list.yaml is used to collect necessary information of secure partition. The manifest tool tools/tfm_parse_manifest_list.py processes it and generates necessary files while building.

Please refer to the *Manifest List* for the format of manifest lists.

Reference configuration example:

```
{
 "description": "TFM Example Partition",
 "manifest": "secure_fw/partitions/example/tfm_example_partition.yaml",
 "conditional": "@TFM_PARTITION_EXAMPLE@",
 "output_path": "partitions/example",
 "version_major": 0,
 "version_minor": 1,
 "pid": 290,
 "linker_pattern": {
  "library_list": [
   "*tfm_*partition_example*"
  ]
 }
}
```

TF-M also supports out-of-tree Secure Partition build where you can have your own manifest lists. Please refer to *Out-of-tree Secure Partition build* for details.

Secure Partition ID Distribution

Every Secure Partition has an identifier (ID). TF-M will generate a header file that includes definitions of the Secure Partition IDs. The header file is <TF-M build folder>generated/interface/include/psa_manifest/pid.h. Each definition uses the name attribute in the manifest as its name

and the value is allocated by SPM.

The Partition ID can be set to a fixed value or omitted to be auto allocated.

```
#define name id-value
```

Table 37: PID table

Secure Partitions	PID Range
TF-M Internal Partitions	0 - 255
PSA and user Partitions	256 - 2999
TF-M test Partitions	3000 - 4999
Firmware Framework test Partitions	5000 - 5999
Reserved	6000 -

Please refer to <TF-M base folder>/tools/tfm_manifest_list.yaml, <TF-M test repo>/test/secure_fw/tfm_test_manifest_list.yaml and <TF-M base folder>/tools/tfm_psa_ff_test_manifest_list.yaml for the detailed PID allocations.

About where to add the definition, please refer to the chapter *Update manifest list*.

RoT Service ID (SID) Distribution

An RoT Service is identified by its RoT Service ID (SID). A SID is a 32-bit number that is associated with a symbolic name in the Secure Partition manifest. The bits [31:12] uniquely identify the vendor of the RoT Service. The remaining bits [11:0] can be used at the discretion of the vendor.

Here is the RoT Service ID table used in TF-M.

Table 38: SID table

Partitions	Vendor ID(20 bits)	Function ID(12 bits)
initial_attestation	0x00000	0x020-0x03F
platform	0x00000	0x040-0x05F
protected_storage	0x00000	0x060-0x06F
internal_trusted_storage	0x00000	0x070-0x07F
crypto	0x00000	0x080-0x09F
firmware_update	0x00000	0x0A0-0x0BF
tfm_secure_client	0x0000F	0x000-0x01F
tfm_ipc_client	0x0000F	0x060-0x07F
tfm_ipc_service	0x0000F	0x080-0x09F
tfm_slih_test_service	0x0000F	0x0A0-0x0AF
tfm_flih_test_service	0x0000F	0x0B0-0x0BF
tfm_ps_test_service	0x0000F	0x0C0-0x0DF
tfm_secure_client_2	0x0000F	0x0E0-0x0FF
tfm_sfn_test_service_1	0x0000F	0x100-0x11F
tfm_sfn_test_service_2	0x0000F	0x120-0x13F
tfm_attest_test_service	0x0000F	0x140-0x15F

RoT Service Stateless Handle Distribution

A Secure partition may include stateless services. They are distinguished and referenced by stateless handles. In manifest, a stateless_handle attribute is set for indexing stateless services. It must be either "auto" or a number in the range [1, 32] in current implementation and may extend. Also the connection-based attribute must be set to false for stateless services.

The indexes of stateless handles are divided into two ranges for different usages. Indexes [1, 16] are assigned to TF-M Secure Partitions. The rest indexes [17, 32] are reserved for any other Secure Partitions, for example Secure Partitions in tf-m-tests and tf-m-extras.

The following table summaries the stateless handle allocation for the TF-M Secure Partitions.

Table 39: Stateless Handle table

Partition name	Stateless Handle
TFM_SP_CRYPTO	1
TFM_SP_PS	2
TFM_SP_ITS	3
TFM_SP_INITIAL_ATTESTATION	4
TFM_SP_FWU	5
TFM_SP_PLATFORM	6

For the indexes of other Secure Partitions, please refer to their manifests or documentations.

stack_size

The stack_size is required to indicate the stack memory usage of the Secure Partition. The value of this attribute must be a decimal or hexadecimal value in bytes. It can also be a build configurable with default value defined in config_base.cmake. The value of the configuration can be overridden to fit different use cases.

heap_size

This attribute is optional. The default value is 0. It indicates the heap memory usage of the Secure Partition. The allowed values are the same as the stack_size.

mmio_regions

This attribute is a list of MMIO region objects which the Secure Partition needs access to. TF-M only supports the named_region current. Please refer to PSA FF for more details about it. The user needs to provide a name macro to indicate the variable of the memory region.

TF-M uses the below structure to indicate a peripheral memory.

```
struct platform_data_t {
  uint32_t periph_start;
  uint32_t periph_limit;
  int16_t periph_ppc_bank;
  int16_t periph_ppc_loc;
};
```

Note: This structure is not expected by TF-M, it's only that the current implementations are using. Other peripherals that need different information to create isolation need to define a different structure with the same name.

Here is an example for it:

```
struct platform_data_t tfm_peripheral_A;
#define TFM_PERIPHERAL_A                (&tfm_peripheral_A)
```

mm_iovec

Memory-mapped iovecs (MM-IOVEC) provides direct mapping of client input and output vectors into the Secure Partition. When this attribute is set to enable, it enables Secure Partitions to use the MM-IOVEC APIs if the framework supports MM-IOVEC.

Using MM-IOVEC provides a memory and runtime optimization for larger buffers, but reduces mitigation for common security vulnerabilities. Please refer to Firmware Framework for M 1.1

Extensions for more details. Whether to use MM-IOVEC depends on the requirements of memory and runtime optimization and security.

Update the Build System

The following changes to the build system are required for the newly added secure partition.

Add a CMakeLists.txt file

Each Secure Partition must have a corresponding CMakeLists.txt, in this case, <TF-M base folder>/secure_fw/partitions/example/CMakeLists.txt, which is the compilation configuration for this Secure Partition.

Here is a reference example for CMakeLists.txt

The CMake file should include the following contents

- Add library tfm_app_rot_partition_example and associated source files.

```
add_library(tfm_app_rot_partition_example STATIC)

target_sources(tfm_app_rot_partition_example
    PRIVATE
        tfm_example_partition.c
)
```

Note: The secure partition must be built as a standalone static library, and the name of the library must follow this pattern, as it affects how the linker script will lay the partition in memory:

  - tfm_psa_rot_partition* in case of a PSA RoT partition
  - tfm_app_rot_partition* in case of an Application RoT partition

---

- Add source files generated by the manifest tool.

```
# The intermedia file defines the partition stack.
target_sources(tfm_app_rot_partition_example
    PRIVATE
        ${CMAKE_BINARY_DIR}/generated/example_partition/auto_generated/intermedia_tfm_
↪example_partition.c
)

# The load info file includes the static data of the partition.
target_sources(tfm_partitions
    INTERFACE
        ${CMAKE_BINARY_DIR}/generated/example_partition/auto_generated/load_info_tfm_
↪example_partition.c
)
```

- Add dependency with manifest tool.

  To make sure the above generated files are up-to-date when the Secure Partition library is built, dependencies between the library and the manifest tool target should be set up.

```
add_dependencies(tfm_app_rot_partition_example manifest_tool)
```

- Link tfm_sprt for the PSA API interfaces.

```
target_link_libraries(tfm_app_rot_partition_example
    PRIVATE
        tfm_sprt
)
```

- Link the Secure Partition library to tfm_partitions so that it can be included in the final image.

```
target_link_libraries(tfm_partitions
    INTERFACE
        tfm_app_rot_partition_example
)
```

Finally, the build of this Secure Partition should be added to <TF-M base folder>/secure_fw/partitions/CMakeLists.txt.

```
add_subdirectory(example)
```

Update the Config System

If the Secure Partition has the build config to enable or disable it, the config option should be added to config systems.

CMake Config

The default value of the config option should be added to the <TF-M base folder>/config/config_base.cmake.

```
set(TFM_PARTITION_EXAMPLE  OFF  CACHE BOOL  "Enable the example partition")
```

Kconfig

A menuconfig should be added to <TF-M base folder>/secure_fw/partitions/example/Kconfig.

```
menuconfig TFM_PARTITION_EXAMPLE
    bool "Enable the Example Partition"
    default n
```

And add it to <TF-M base folder>/secure_fw/partitions/Kconfig

```
rsource ``example/Kconfig``
```

Note: The Secure Partition building should be skipped if it is not enabled. This should be done by adding the following code at the begining of its CMakeLists.txt

```
if (NOT TFM_PARTITION_EXAMPLE)
    return()
endif()
```

Implement the RoT services

To implement RoT services, the partition needs a source file which contains the implementations of the services, as well as the partition entry point. The user can create this source file under <TF-M base folder>/secure_fw/partitions/example/tfm_example_partition.c.

As an example, the RoT service with SID **ROT_A** will be implemented.

Entry point for IPC Model Partitions

This function must have a loop that repeatedly waits for input signals and then processes them, following the Secure Partition initialization.

```
#include "psa_manifest/tfm_example.h"
#include "psa/service.h"

void tfm_example_main(void)
{
    psa_signal_t signals = 0;

    /* Secure Partition initialization */
    example_init();

    /*
     * Continually wait for one or more of the partition's RoT Service or
     * interrupt signals to be asserted and then handle the asserted
     * signal(s).
     */
    while (1) {
        signals = psa_wait(PSA_WAIT_ANY, PSA_BLOCK);
        if (signals & ROT_A_SIGNAL) {
            rot_A();
        } else {
            /* Should not come here */
            psa_panic();
        }
    }
}
```

Entry init for SFN Model Partitions

In the SFN model, the Secure Partition consists of one optional initialization function, which is declared as the entry_init symbol as mentioned in section *Add manifest*. After initialization, the entry_init function returns the following values:

- Return PSA_SUCCESS if initialization succeeds.
- Return PSA_SUCCESS if initialization is partially successful, and you want some SFNs to receive messages. RoT Services that are non-operational must respond to connection requests with PSA_ERROR_CONNECTION_REFUSED.
- Return an error status if the initialization failed, and no SFNs within the Secure Partition must be called.

Service implementation for IPC Model

The service is implemented by the rot_A() function, which is called upon an incoming signal. This implementation is up to the user, however an example service has been included for reference. The following example sends a message "Hello World" when called.

```
#include "psa_manifest/tfm_example.h"
#include "psa/service.h"

/* Some other type of services. */
#define SOME_ROT_A_SERVICE_TYPE                (1)

static void rot_A(void)
{
    const int BUFFER_LEN = 32;
    psa_msg_t msg;
    int i;
    uint8_t rec_buf[BUFFER_LEN];
    uint8_t send_buf[BUFFER_LEN] = "Hello World";

    psa_get(ROT_A_SIGNAL, &msg);
```

(continues on next page)

(continued from previous page)

```
    switch (msg.type) {
    case PSA_IPC_CONNECT:
    case PSA_IPC_DISCONNECT:
        /*
         * This service does not require any setup or teardown on connect
         * or disconnect, so just reply with success.
         */
        psa_reply(msg.handle, PSA_SUCCESS);
        break;
    default:
        /* Handling services requested by psa_call. */
        if (msg.type == PSA_IPC_CALL) {
            for (i = 0; i < PSA_MAX_IOVEC; i++) {
                if (msg.in_size[i] != 0) {
                    psa_read(msg.handle, i, rec_buf, BUFFER_LEN);
                }
                if (msg.out_size[i] != 0) {
                    psa_write(msg.handle, i, send_buf, BUFFER_LEN);
                }
            }
            psa_reply(msg.handle, PSA_SUCCESS);
        } else if (msg.type == SOME_ROT_A_SERVICE_TYPE) {
            /* Operations for SOME_ROT_A_SERVICE_TYPE */
        } else {
            /* Invalid type for this Secure Partition. */
            return PSA_ERROR_PROGRAMMER_ERROR;
        }
    }
}
```

Service implementation for SFN Model

SFN model consists of a set of Secure Functions (SFN), one for each RoT Service. The connection, disconnection and request messages do not cause a Secure Partition signal to be asserted for SFN Secure Partitions. Instead, the Secure Function (SFN) for the RoT Service is invoked by the framework, with the message details provided as a parameter to the SFN. To add a secure function (SFN) to process messages for each RoT Service, each SFN will have following prototype.

```
psa_status_t <<name>>_sfn(const psa_msg_t *msg);
```

A connection-based example service has been included for reference which sends a message “Hello World” when called.

```
#include "psa_manifest/tfm_example.h"
#include "psa/service.h"

/* Some other type of services. */
#define SOME_ROT_A_SERVICE_TYPE              (1)

psa_status_t rot_a_sfn(const psa_msg_t *msg)
{
    const int BUFFER_LEN = 32;
    int i;
    uint8_t rec_buf[BUFFER_LEN];
    uint8_t send_buf[BUFFER_LEN] = "Hello World";

    switch (msg->type) {
    case PSA_IPC_CONNECT:
    case PSA_IPC_DISCONNECT:
        /*
```

(continues on next page)

(continued from previous page)

```
        * This service does not require any setup or teardown on connect
        * or disconnect, so just reply with success.
        */
       return PSA_SUCCESS;
   default:
       /* Handling services requested by psa_call. */
       if (msg->type == PSA_IPC_CALL) {
           for (i = 0; i < PSA_MAX_IOVEC; i++) {
               if (msg->in_size[i] != 0) {
                   psa_read(msg->handle, i, rec_buf, BUFFER_LEN);
               }
               if (msg.->out_size[i] != 0) {
                   psa_write(msg->handle, i, send_buf, BUFFER_LEN);
               }
           }
           return PSA_SUCCESS;
       } else if (msg->type == SOME_ROT_A_SERVICE_TYPE) {
           /* Operations for SOME_ROT_A_SERVICE_TYPE */
       } else {
           /* Invalid type for this Secure Partition. */
           return PSA_ERROR_PROGRAMMER_ERROR;
       }
   }
}
```

Test suites and test partitions

A regression test suite can be added to verify whether the new secure partition works as expected. Refer to Adding TF-M Regression Test Suite for the details of adding a regression test suite.

Some regression tests require a dedicated RoT service. The implementations of the RoT service for test are similar to secure partition addition. Refer to Adding partitions for regression tests to get more information.

Out-of-tree Secure Partition build

TF-M supports out-of-tree Secure Partition build, whose source code folders are maintained outside TF-M repo. Developers can configure TFM_EXTRA_MANIFEST_LIST_FILES and TFM_EXTRA_PARTITION_PATHS in build command line to include out-of-tree Secure Partitions.

- TFM_EXTRA_MANIFEST_LIST_FILES

  A list of the absolute path(s) of the manifest list(s) provided by out-of-tree Secure Partition(s). Use semicolons ; to separate multiple manifest lists. Wrap the multiple manifest lists with double quotes.

- TFM_EXTRA_PARTITION_PATHS

  A list of the absolute directories of the out-of-tree Secure Partition source code folder(s). TF-M build system searches CMakeLists.txt of partitions in the source code folder(s). Use semicolons ; to separate multiple out-of-tree Secure Partition directories. Wrap the multiple directories with double quotes.

A single out-of-tree Secure Partition folder can be organized as the figure below.

```
secure partition folder
        CMakeLists.txt
        manifest_list.yaml
        out_of_tree_partition_manifest.yaml
        source code
```

In the example above, TFM_EXTRA_MANIFEST_LIST_FILES and TFM_EXTRA_PARTITION_PATHS in the build command can be configured as listed below.

```
-DTFM_EXTRA_MANIFEST_LIST_FILES=<Absolute-path-sp-folder/manifest_list.yaml>
-DTFM_EXTRA_PARTITION_PATHS=<Absolute-path-sp-folder>
```

Multiple out-of-tree Secure Partitions can be organized in diverse structures. For example, multiple Secure Partitions can be maintained under the same directory as shown below.

```
top-level folder
        Partition 1
                CMakeLists.txt
                partition_1_manifest.yaml
                source code
        Partition 2
                ...
        Partition 3
                ...
        manifest_list.yaml
        Root CMakeLists.txt
```

In the example above, a root CMakeLists.txt includes all the partitions' CMakLists.txt, for example via add_subdirectory(). The manifest_list.yaml lists all partitions' manifest files. TFM_EXTRA_MANIFEST_LIST_FILES and TFM_EXTRA_PARTITION_PATHS in build command line can be configured as listed below.

```
-DTFM_EXTRA_MANIFEST_LIST_FILES=<Absolute-path-top-level-folder/manifest_list.yaml>
-DTFM_EXTRA_PARTITION_PATHS=<Absolute-path-top-level-folder>
```

Alternatively, out-of-tree Secure Partitions can be separated in different folders.

```
partition 1 folder                  partition 2 folder
        CMakeLists.txt                      CMakeLists.txt
        manifest_list.yaml                  manifest_list.yaml
        partition_1_manifest.yaml               partition_2_manifest.yaml
        source code                     source code
```

In the example above, each Secure Partition manages its own manifest files and CMakeLists.txt. TFM_EXTRA_MANIFEST_LIST_FILES and TFM_EXTRA_PARTITION_PATHS in build command line can be configured as listed below. Please note those input shall be wrapped with double quotes.

```
-DTFM_EXTRA_MANIFEST_LIST_FILES="<Absolute-path-part-1-folder/manifest_list.yaml>;
↪<Absolute-path-part-2-folder/manifest_list.yaml>"
-DTFM_EXTRA_PARTITION_PATHS="<Absolute-path-part-1-folder>;<Absolute-path-part-2-folder>"
```

Note: Manifest list paths in TFM_EXTRA_MANIFEST_LIST_FILES do NOT have to be one-to-one mapping to Secure Partition directories in TFM_EXTRA_PARTITION_PATHS. The orders don't matter either.

TFM_EXTRA_MANIFEST_LIST_FILES and TFM_EXTRA_PARTITION_PATHS can be configurated in multiple extra sources. It is recommended to use CMake list APPEND method to avoid unexpected override.

Further Notes

- In the IPC model, Use PSA FF proposed memory accessing mechanism. SPM provides APIs and checking between isolation boundaries, a free accessing of memory can cause program panic.

- In the IPC model, the memory checking inside partition runtime is unnecessary. SPM handles the checking while memory accessing APIs are called.
- In the IPC model, the client ID had been included in the message structure and secure partition can get it when calling psa_get() function. The secure partition does not need to call tfm_core_get_caller_client_id() to get the caller client ID anymore.
- In the IPC model, SPM will check the security policy and partition dependence between client and service. So the service does not need to validate the secure caller anymore.

Reference

PSA Firmware Framework specification
Firmware Framework for M 1.1 Extensions

Copyright (c) 2019-2022, Arm Limited. All rights reserved. Copyright (c) 2022 Cypress Semiconductor Corporation (an Infineon company) or an affiliate of Cypress Semiconductor Corporation. All rights reserved.

TF-M Manifest Tool User Guide

This document describes the TF-M manifest tool and its usage. The target audiences are mainly platform integrators and Secure Partition developers.

This document assumes that audiences have knowledge about the terminologies defined in PSA Firmware Framework (FF-M) v1.0[1] and the FF-M v1.1 extensions[2], such as Secure Partition and manifests.

Introduction

The TF-M manifest tool is a python script which is used to parse Secure Partition manifests and generate source header files defined by FF-M[Page 1312, 1] such as psa_manifest/pid.h and psa_manifest/sid.h. It also generates TF-M specific files for building.

In the TF-M build system, the manifest tool is invoked during building automatically. Arguments are passed through CMake variables and they are customizable. See *Usage in TF-M Build System* for more details.

It can be also used as a standalone tool.

Arguments

The tool takes 5 arguments at most.

```
tfm_parse_manifest_list.py [-h] -o out_dir
                           -m manifest list [manifest list ...]
                           -f file-list [file-list ...]
                           -c config-files [config-files ...]
                           [-q]
```

-o/–outdir

Required.

The directory to hold the output files.

-m/–manifest-lists

[1] FF-M v1.0 Specification
[2] FF-M v1.1 Extension

Required. At least one item must be provided.

A list of TF-M Secure Partition manifest lists which contain one or more Secure Partition manifests in each. See *Manifest List* for details.

-f/–file-list

Required. At least one item must be provided.

A list of files that describe what files the tool should generate. See *Generated File List* for details.

-c/–config-files

Required. At least one item must be provided.

A list of header files which contain build configurations such as isolation level and Secure Partition enabled status. See *Configuration Header File* for details.

-q/–quiet

Optional.

Reduces log messages, only warnings and errors will be printed.

Manifest List

A manifest list is a YAML[3] file that describes a list of Secure Partition manifests. Refer to the TF-M default manifest list[4] as an example.

Each manifest list must contain a `manifest_list` attribute which collects the descriptions of Secure Partition manifests. Following are the supported attributes of in the manifest lists.

- `description`

  Required.

  Descriptive information for the tool to refer to the Secure Partition, for example in logs.

- `manifest`

  Required.

  The manifest file of the Secure Partition. Both absolute path and relative path to the manifest list are supported. Environment variables are supported.

- `output_path`

  Optional.

  The directory to hold the Secure Partition specific output files. See *Generated File List* for more details.

  Both absolute path and relative path to the directory specified by `-o/--outdir` are supported. Environment variables are supported.

  It is set to the directory specified by `-o/--outdir` if omitted.

- `conditional`

  Optional.

  The configuration to enable or disable this Secure Partition. The value must be defined in one of the *Configuration Header File*. If it is omitted, the Secure Partition is always enabled.

- `pid`

  Optional.

  The Secure Partition ID (PID).

[3] YAML
[4] TF-M manifest list

For non-test purpose Secure Partitions, it is required. See *Adding Secure Partition* for the PID allocations.

For test purpose Secure Partitions, this attribute is optional. The manifest tool assigns one for the Secure Partition. The value is not guaranteed to the same for different builds.

- linker_pattern

  Optional.

  The information for linker to place the symbols of the Secure Partition. It is only required if you are using the linker scripts provided by TF-M. Each Secure Partition is expected to be built as a library. The name of the library must follow the format of tfm_<type>_partition_<name>. The valid value for <type> is [psa_rot, app_rot] corresponding to the type of the Secure Partitions. The <name> is any string to distinguish the Secure Partition from others.

  Supported patterns are:

  - library_list, must be *tfm_*partition_<name>.*.
  - object_list

    Any object files containing symbols belonging to the Secure Partition that are not included in the Secure Partitions library.

- non_ffm_attributes

  Optional.

  TF-M defines some special manifest attributes for TF-M dedicated Secure Partitions. Those special attributes are not compliant to FF-M. Using the specific attributes requires explicit registration by adding them to this non_ffm_attributes. The purpose is to ensure that developers are aware of the compliance issue. The manifest tool will report errors if unregistered Non-FFM attributes are detected. This attribute is for TF-M specific Secure Partitions and using TF-M-specific attributes is not encouraged.

Generated File List

A generated file list is a YAML file that describes the files to be generated by the manifest tool. Refer to TF-M default generated file list[5] as an example.

Each one must contain a file_list attribute which collects the files to generate. Each item in the file_list must contain the following attributes.

- template

  This attribute is the file path of a Jinja2[6] template. The TF-M manifest tool uses Jinja2 template engine for file generations. It can be a relative path to TF-M root directory or an absolute path. Environment variables are supported.

- output

  The output file of the template. Both absolute path and relative path to the directory specified by -o/--outdir are supported. Environment variables are supported.

The tfm_generated_file_list.yaml is essential to build TF-M.

There are several files that are required for each Secure Partition, so they are not in any generated file lists since one template generates multiple files.

- psa_manifest/<manifestfilename>.h

  manifestfilename is the file name of the manifest. This file contains internal definitions for the Secure Partition implementation, such as RoT Service signals and Secure Functions. Refer to FF-M[1] for more details. The corresponding template is manifestfilename.template

[5] TF-M generated file list
[6] Jinja2

- intermedia_<manifestfilename>.c

  TF-M specific, which holds the stacks of Secure Partitions. This file must be built with the Secure Partition libraries. The corresponding template is partition_intermedia.template.

- load_info_<manifestfilename>.c

  TF-M specific, which contains the load information of Secure Partitions. This file must be built with the TF-M SPM library. The corresponding template is partition_load_info.template.

These files are generated to output_path specified by each Secure Partition in the manifest lists.

Configuration Header File

The format of each configuration item must be

```
#define CONFIG_NAME    VALUE
```

The following format is also supported for boolean type configurations.

```
#define CONFIG_NAME
```

The configurations can be divided into two categories.

- Generic configurations:
  - PSA_FRAMEWORK_ISOLATION_LEVEL

    The isolation level, required. Valid values are [1, 2, 3].

  - CONFIG_TFM_SPM_BACKEND

    The backend of SPM, required. Valid values are [SFN, IPC]. See *SPM backends* for details of backends.

- Secure Partition enablement configurations

  Configurations used to enable or disable Secure Partitions. The configuration names must match the values of conditional attributes in *Manifest List*. Valid values are [0, 1]. It's optional for a Secure Partition which does not have the conditional attribute.

The configurations can be split to multiple files corresponding to the multiple manifest lists.

Usage in TF-M Build System

In the TF-M build system, the manifest tool is invoked during building automatically. The arguments can be customized by altering the CMake configurations.

The manifest lists are passed to the manifest tool via the TFM_MANIFEST_LIST CMake configuration. The default value is the tfm_manifest_list.yaml. It can be overridden or appended with other manifest lists.

Corresponding manifest lists of test Secure Partitions are appended if either TF-M regression or PSA compliance tests are enabled.

The generated file lists are passed via GENERATED_FILE_LISTS. It can be also overridden or appended with other file lists.

The -q argument is appended if PARSE_MANIFEST_QUIET_FLAG is enabled.

Paths in manifest lists and generated file lists can have CMake variables as long as the are absolute paths. The lists then must be processed by the CMake command configure_file()[7] before passing to the manifest tool.

The configuration header file is generated by the build system automatically.

[7] CMake configure_file()

References

Copyright (c) 2022, Arm Limited. All rights reserved. Copyright (c) 2022 Cypress Semiconductor Corporation (an Infineon company) or an affiliate of Cypress Semiconductor Corporation. All rights reserved.

Copyright (c) 2020-2022, Arm Limited. All rights reserved.

How to build TF-M

Follow the *Build instructions*.

How to export files for building non-secure applications

Explained in the *Build instructions*.

How to add a new platform

Porting TF-M to a New Hardware contains guidance on how to add a new platform.

How to integrate another OS

OS migration to Armv8-M platforms

To work with TF-M on Armv8-M platforms, the OS needs to support the Armv8-M architecture and, in particular, it needs to be able to run in the non-secure world. More information about OS migration to the Armv8-M architecture can be found in the *OS requirements*. Depending upon the system configuration this may require configuring drivers to use appropriate address ranges.

Interface with TF-M

The files needed for the interface with TF-M are exported at the <install_dir>/interface path. The NS side is only allowed to call TF-M secure functions (veneers) from the NS Thread mode.

TF-M interface header files are exported in <install_dir>/interface/include directory. For example, the Protected Storage (PS) service PSA API is declared in the file <install_dir>/interface/include/psa/protected_storage.h.

TF-M also exports a reference implementation of PSA APIs for NS clients in the <install_dir>/interface/src.

On Armv8-M TrustZone based platforms, NS OS uses tfm_ns_interface_dispatch() to integrate with TF-M implementation of PSA APIs. TF-M provides a reference implementation of this function for RTOS and bare metal use cases. RTOS implementation of tfm_ns_interface_dispatch() (provided in interface\src\os_wrapper\tfm_ns_interface_rtos.c) uses mutex to provide multithread safety. Mutex wrapper functions defined in interface/include/os_wrapper/mutex.h are expected to be provided by NS RTOS. When reference RTOS implementation of dispatch function is used NS application should call tfm_ns_interface_init() function before first PSA API call. Bare metal implementation tfm_ns_interface_dispatch() (provided in interface\src\os_wrapper\tfm_ns_interface_bare_metal.c) does not provide multithread safety and does not require implementation of mutex interface. If needed, instead of using reference implementation, NS application may provide its own implementation of tfm_ns_interface_dispatch() function.

TF-M provides a reference implementation of NS mailbox on multi-core platforms, under folder interface/src/multi_core. See *Mailbox design* for TF-M multi-core mailbox design.

Interface with non-secure world regression tests

A non-secure application that wants to run the non-secure regression tests needs to call the tfm_non_secure_client_run_tests(). This function is exported into the header file test_framework_integ_test.h inside the <build_dir>/install folder structure in the test specific files, i.e. <build_dir>/install/export/tfm/test/inc. The non-secure regression tests are precompiled and delivered as a static library which is available in <build_dir>/install/export/tfm/test/lib, so that the non-secure application needs to link against the library to be able to invoke the tfm_non_secure_client_run_tests() function. The PS non-secure side regression tests rely on some OS functionality e.g. threads, mutexes etc. These functions comply with CMSIS RTOS2 standard and have been exported as thin wrappers defined in os_wrapper.h contained in <build_dir>/install/export/tfm/test/inc. OS needs to provide the implementation of these wrappers to be able to run the tests.

NS client Identification

The NS client identification (NSID) is specified by either SPM or NSPE RTOS. If SPM manages the NSID (default option), then the same NSID (-1) will be used for all connections from NS clients. For the case that NSPE RTOS manages the NSID and/or different NSIDs should be used for different NS clients. See *Non-secure Client Extension Integration Guide*.

Non-secure interrupts

Non-secure interrupts are allowed to preempt Secure thread mode. With the current implementation, a NSPE task can spoof the identity of another NSPE task. This is an issue only when NSPE has provisions for task isolation. Note, that AIRCR.PRIS is still set to restrict the priority range available to NS interrupts to the lower half of available priorities so that it wouldn't be possible for any non-secure interrupt to preempt a higher-priority secure interrupt.

Secure interrupts and scheduling

To ensure correct operation in the general case, the secure scheduler is not run after handling a secure interrupt that pre-empted the NSPE. On systems with specific constraints, it may be desirable to run the scheduler in this situation, which can be done by setting CONFIG_TFM_SCHEDULE_WHEN_NS_INTERRUPTED to 1. This could be done if the NSPE is known to be a simple, single-threaded application or if non-secure interrupts cannot pre-empt the SPE, for example.

Integration with non-Cmake systems

Generated Files

Files that are derived from PSA manifests are generated at build-time by cmake. For integration with systems that do no use cmake, the files must be generated manually.

The tools/tfm_parse_manifest_list.py script can be invoked manually. Some arguments will be needed to be provided. Please refer to tfm_parse_manifest_list.py --help for more details.

Some variables are used in the template files, these will need to be set in the environment before the script will succeed when the script is not run via cmake.

Copyright (c) 2017-2022, Arm Limited. All rights reserved. Copyright (c) 2023 Cypress Semiconductor Corporation (an Infineon company) or an affiliate of Cypress Semiconductor Corporation. All rights reserved.

Design Documents

Secure boot

BL1 Immutable bootloader

Author
Raef Coles

Organization
Arm Limited

Contact
raef.coles@arm.com

Introduction

Some devices that use TF-M will require initial boot code that is stored in ROM. There are a variety of reasons that this might happen:

- The device cannot access flash memory without a driver, so needs some setup to be done before main images on flash can be booted.
- The device has no on-chip secure flash, and therefore cannot otherwise maintain a tamper-resistant root of trust.
- The device has a security model that requires an immutable root of trust

Henceforth any bootloader stored in ROM will be referred to as BL1, as it would necessarily be the first stage in the boot chain.

TF-M provides a reference second-stage flash bootloader BL2, in order to allow easier integration. This bootloader implements all secure boot functionality needed to provide a secure chain of trust.

A reference ROM bootloader BL1 has now being added with the same motivation - allowing easier integration of TF-M for platforms that do not have their own BL1 and require one.

BL1 Features and Motivations

The reference ROM bootloader provides the following features:

- A split between code being stored in ROM and in other non-volatile memory.
  - This can allow significant cost reduction in fixing bugs compared to ROM-only bootloaders.
- A secure boot mechanism that allows upgrading the next boot stage (which would usually be BL2).
  - This allows for the fixing of any bugs in the BL2 image.
  - Alternately, this could allow the removal of BL2 in some devices that are constrained in flash space but have ROM.
- A post-quantum resistant asymmetric signature scheme for verifying the next boot stage image.
  - This can allow devices to be securely updated even if attacks involving quantum computers become viable. This could extend the lifespans of devices that might be deployed in the field for many years.
- A mechanism for passing boot measurements to the TF-M runtime so that they can be attested.
- Tooling to create and sign images.
- Fault Injection (FI) and Differential Power Analysis (DPA) mitigations.

BL1_1 and BL1_2 split bootloaders

BL1 is split into two distinct boot stages, BL1_1 which is stored in ROM and BL1_2 which is stored in other non-volatile storage. This would usually be either trusted or untrusted flash, but on

platforms without flash memory can be OTP. As BL1_2 is verified against a hash stored in OTP, it is immutable after provisioning even if stored in mutable storage.

Bugs in ROM bootloaders usually cannot be fixed once a device is provisioned / in the field, as ROM code is immutable the only option is fixing the bug in newly manufactured devices.

However, it can be very expensive to change the ROM code of devices once manufacturing has begun, as it requires changes to the photolithography masks that are used to create the device. This cost varies depending on the complexity of the device and of the process node that it is being fabricated on, but can be large, both in engineering time and material/process costs.

By placing the majority of the immutable bootloader in other storage, we can mitigate the costs associated with changing ROM code, as a new BL1_2 image can be used at provisioning time with minimal changeover cost. BL1_1 contains a minimal codebase responsible mainly for the verification of the BL1_2 image.

The bootflow is as follows. For simplicity this assumes that the boot stage after BL1 is BL2, though this is not necessarily the case:

1) BL1_1 begins executing in place from ROM
2) BL1_1 copies BL1_2 into RAM
3) BL1_1 verifies BL1_2 against the hash stored in OTP
4) BL1_1 jumps to BL1_2, if the hash verification has succeeded
5) BL1_2 copies the primary BL2 image from flash into RAM
6) BL1_2 verifies the BL2 image using asymmetric cryptography
7) If verification fails, BL1_2 repeats 5 and 6 with the secondary BL2 image
8) BL1_2 jumps to BL2, if either image has successfully verified

---

Note: The BL1_2 image is not encrypted, so if it is placed in untrusted flash it will be possible to read the data in the image.

Some optimizations have been made specifically for the case where BL1_2 has been stored in OTP:

OTP can be very expensive in terms of chip area, though new technologies like antifuse OTP decrease this cost. Because of this, the code size of BL1_2 has been minimized. Code-sharing has been configured so that BL1_2 can call functions stored in ROM. Care should be taken that OTP is sized such that it is possible to include versions of the functions used via code-sharing, in case the ROM functions contain bugs, though less space is needed than if all code is duplicated as it is assumed that most functions will not contain bugs.

As OTP memory frequently has low performance, BL1_2 is copied into RAM before it it is executed. It also copies the next image stage into RAM before authenticating it, which allows the next stage to be stored in untrusted flash. This requires that the device have sufficient RAM to contain both the BL1_2 image and the next stage image at the same time. Note that this is done even if BL1_2 is located in XIP-capable flash, as it both allows the use of untrusted flash and simplifies the image upgrade logic.

Note: BL1_2 enables TF-M to be used on devices that contain no secure flash, though the ITS service will not be available. Other services that depend on ITS will not be available without modification.

Secure boot / Image upgrade mechanism

BL1_2 verifies the authenticity of the next stage image via asymmetric cryptography, using a public key that is provisioned into OTP.

BL1_2 implements a rollback protection counter in OTP, which is used to prevent the next stage image being downgraded to a less secure version.

BL1_2 has two image slots, which allows image upgrades to be performed. The primary slot is always booted first, and then if verification of this fails (either due to an invalid signature or due to a version lower than the rollback protection counter) the secondary slot is then booted (subject to the same checks).

BL1_2 contains no image upgrade logic, in order for OTA of the next stage image to be implemented, a later stage in the system must handle downloading new images and placing them in the required slot.

Post-Quantum signature verification in BL1_2

BL1_2 uses a post-quantum asymmetric signature scheme to verify the next stage. The scheme used is Leighton-Michaeli Signatures (henceforth LMS). LMS is standardised in NIST SP800-208 and IETF RFC8554.

LMS is a stateful-hash signature scheme, meaning that:

1) It is constructed from a cryptographic hash function, in this case SHA256.
    - This function can be accelerated by existing hardware accelerators, which can make LMS verification relatively fast compared to other post-quantum signature schemes that cannot be accelerated in hardware yet.
2) Each private key can only be used to sign a certain number of images.
    - BL1_2 uses the SHA256_H10 parameter set, meaning each key can sign 1024 images.

The main downside, the limited amount of possible signatures, can be mitigated by limiting the amount of image upgrades that are done. As BL2 is often currently not upgradable, it is not anticipated that this limit will be problematic. If BL1 is being used to directly boot a TF-M/NS combined image, the limit is more likely to be problematic, and care should be taken to examine the likely update amount.

LMS public keys are 32 bytes in size, and LMS signatures are 1912 bytes in size. The signature size is larger than some asymmetric schemes, though most devices should have enough space in flash to accommodate this.

The main upside of LMS, aside from the security against attacks involving quantum computers, is that it is relatively simple to implement. The software implementation that is used by BL1 is ~3KiB in size, which is considerably smaller than the corresponding RSA implementation which is at least 6.5K. This simplicity of implementation is useful to avoid bugs.

BL1 will use MbedTLS as the source for its implementation of LMS.

Note: As of the time of writing, the LMS code is still in the process of being merged into MbedTLS, so BL1 currently does not support asymmetric verification of the next boot stage. Currently, the next boot stage is hash-locked, so cannot be upgraded.

The Github pull request for LMS can be found here

BL1 boot measurements

BL1 outputs boot measurements in the same format as BL2, utilising the same shared memory area. These measurements can then be included in the attestation token, allowing the attestation of the version of the boot stage after BL1.

BL1 tooling

Image signing scripts are provided for BL1_1 and BL1_2. While the script is named `create_bl2_img.py`, it can be used for any next stage image.

- bl1/bl1_1/scripts/create_bl1_2_img.py
- bl1/bl1_2/scripts/create_bl2_img.py

These sign (and encrypt in the case of create_bl2_img.py) a given image file and append the required headers.

BL1 FI and DPA mitigations

BL1 reuses the FI countermeasures used in the TF-M runtime, which are found in lib/fih/.

BL1 implements countermeasures against fault injection. The functions with these countermeasures are found in bl1/bl1_1/shared_lib/util.c

bl_fih_memeql tests if memory regions have the same value

- It inserts random delays to improve resilience to FIH attacks
- It performs loop integrity checks
- It uses FIH constructs

Using BL1 on new platforms

New platforms must define the following macros in their region_defs.h:

- BL1_1_HEAP_SIZE
- BL1_1_STACK_SIZE
- BL1_2_HEAP_SIZE
- BL1_2_STACK_SIZE
- BL1_1_CODE_START
- BL1_1_CODE_LIMIT
- BL1_1_CODE_SIZE
- BL1_2_CODE_START
- BL1_2_CODE_LIMIT
- BL1_2_CODE_SIZE
- PROVISIONING_DATA_START
- PROVISIONING_DATA_LIMIT
- PROVISIONING_DATA_SIZE

The PROVISIONING_DATA_* defines are used to locate where the data to be provisioned into OTP can be found. These are required as the provisioning bundle needs to contain the entire BL1_2 image, usually >= 8KiB in size, which is too large to be placed in the static data area as is done for all other dummy provisioning data. On development platforms with reprogrammable ROM, this is often placed in unused ROM. On production platforms, this should be located in RAM and then filled with provisioning data. The format of the provisioning data that should be located in the PROVISIONING_DATA_* region can be found in bl1/bl1_1/lib/provisioning.c in the struct bl1_assembly_and_test_provisioning_data_t

If the platform is storing BL1_2 in flash, it must set BL1_2_IMAGE_FLASH_OFFSET to the flash offset of the start of BL1_2.

The platform must also implement the HAL functions defined in the following headers:

- bl1/bl1_1/shared_lib/interface/trng.h
- bl1/bl1_1/shared_lib/interface/crypto.h
- bl1/bl1_1/shared_lib/interface/otp.h

If the platform integrates a CryptoCell-312, then it can reuse the existing implementation.

BL1 Testing

New tests have been written to test both the HAL implementation, and the integration of those functions for verifying images. These tests are stored in the tf-m-tests repository, under the test/bl1/ directory, and further subdivided into BL1_1 and BL1_2 tests.

Copyright (c) 2022-2023, Arm Limited. All rights reserved.

Rollback protection in TF-M secure boot

Author
Tamas Ban

Organization
Arm Limited

Contact
Tamas Ban <tamas.ban@arm.com>

Anti-rollback protection

The goal of anti-rollback protection is to prevent downgrading of the device to an older version of its software, which has been deprecated due to security concerns.

Elements of software upgrade

During a software upgrade the following entities are involved:

- Boot loader
- Manifest data: Metadata of the software image: size, version, hash, signature, etc.
- Software image: binary data, elf, etc.

Validation of new image

Boot loader is responsible to authenticate the new image according to the required policies and decide whether the new image is fulfilling all the requirements. Boot loader verifies the image integrity (hash calculation) and the origination (signature validation), and might verify other requirements as well. If the new image is successfully authenticated then the boot loader is in charge to do the necessary steps (load to execute address, etc.) to enable the new image to be executed. During the validation process the image and the manifest data is checked.

Manifest format in MCUBoot

MCUBoot has a custom manifest format, which is composed of two parts:

- Image header: Prepended to the beginning of the image. It is integrity protected:
- TLV section: Appended to the end of the image. It is not integrity protected:
- Hash of public key, hash of image, signature of image

```
struct image_header {
    uint32_t ih_magic;
    uint32_t ih_load_addr;
    uint16_t ih_hdr_size;
    uint16_t _pad1;
    uint32_t ih_img_size;
```

(continues on next page)

(continued from previous page)

```
    uint32_t ih_flags;
    struct image_version ih_ver;
    uint32_t _pad2;
};
```

Security counter

The aim of a security counter is to have an independent (from the image version) counter in the image manifest to ensure anti-rollback protection. During software release the value of this counter must be increased if a security flaw was fixed in the current version. Later when this image is installed on the device then it is not allowed to go back to earlier versions. It is beneficial to handle this counter independently from image main version number:

- It does not need to increase with each software release
- It makes it possible to do software downgrade to some extent: if the security counter has the same value in the older image then it is accepted.
- If the boot loader verifies multiple images then these can be handled independently.

However, as an alternative solution the image version number also could serve as the security counter of the image. Even the version number itself could be checked during the anti-rollback verification or the value of the security counter could be derived from the image main version. It is the responsibility of the software issuer to define which policy to apply.

Implementation of security counter

The value of the security counter is a security critical data. Any change in its value has a security implication. Therefore it must be in the integrity protected part of the image manifest. Because the image header is almost fully utilized (few unused fields) and the change of image header structure could lead to compatibility issues between boot loader and runtime software, it is proposed to extend the integrity protection to some part of the TLV section. One of the unused fields in the image header can be used to store the size of the integrity protected area of the TLV section. This is necessary to know how to calculate properly the image hash and signature. With this extension of the integrity protected area other attributes that require integrity protection can easily be added to the image manifest.

Trusted non-volatile (NV) counters

The Trusted Base System Architecture (TBSA-M) defines non-volatile (NV) counters or version counters as a counter with the following properties:

- It must only be possible to increment a version counter through a Trusted access.
- It must only be possible to increment a version counter. It must not be possible to decrement it.
- When a version counter reaches its maximum value, it must not roll over, and no further changes must be possible.
- A version counter must be non-volatile, and the stored value must survive a power down period up to the lifetime of the device.

Trusted non-volatile counters can be used to store the value of security counters per updatable software image. Ideally all independently updatable software images should have a separate security counter. In current TF-M implementation the BL2 is not updatable and the secure and non-secure images are updated together as a single binary. Therefore, one counter is enough to implement rollback protection. In future the secure and non-secure binaries will be handled independently; at that time the introduction of a second counter will be necessary.

Currently the NV counters can be manipulated through the interface described in `tfm_plat_nv_counters.h`.

NV counters and anti-rollback protection

Trusted non-volatile counters might not be supported by a hardware platform. In this case anti-rollback protection might still be feasible.

The device threat model needs to consider the following aspects:

- What is the trust level of the boot loader towards the active software
- If the boot loader cannot protect the anti-rollback mechanism from the secure image then the following threat is unmitigated: The content of the memory area which holds the active image could be replaced with a valid but an older version of the software with software only attack, i.e.: remote code execution.
- If the boot loader does not trust the loaded image at all then security counter must have a copy in NV counter area.
- Another aspect to consider is where the active image is stored
- Trusted memory: i.e. on-chip flash (eFlash). The threat that a malicious actor can modify the content of trusted memory with HW attack is out of scope for the current implementation. It is assumed that if an active image and related manifest data is stored in trusted memory then the included security counter cannot be compromised.
- Untrusted memory: i.e. external (off-chip) storage, where malicious actor can physically access it so it is possible to modify its content, therefore the value of included security counter is unreliable.

If the active image is stored in untrusted storage and it is feasible to modify its content (i.e. replace the whole image to an older version including corresponding manifest) then the value of security counter must be copied to the NV counter area. During software validation the boot loader must compare the new image's security counters against the security counter stored in non-volatile counters.

If the active image is stored in trusted memory and boot loader trusts in the active software then it is not mandatory to store the security counter to non-volatile counter area. During software validation the boot loader is allowed to compare the new image's security counters against active image's security counter.

The evaluation of trusted and untrusted memory must be done per target platform during threat modelling.

For the most robust implementation the following principles should be applied:

- Always use NV counters for storing security counter value if supported by the hardware.
- Each software stage must not be able to decrease their corresponding NV counter.

Boot flow with anti-rollback protection

During software upgrade as part of the image validation process the new and active image security counters must be compared. The new image can be accepted if its security counter has a greater or equal value than the active image security counter. From where to extract the active image security counter it can be platform dependent. It might read out directly from active image manifest data (only if it is in trusted memory) or the corresponding non-volatile counter is read.

If non-volatile counters are used to save security counters then their value must be updated:

- If the boot loader does not support to revert previous images (just overwrites the previously active image with the new one) in case of faulty update then the non-volatile counter can be updated to be equal with the new image security counter after successful authentication of the new image.

- If revert is supported then non-volatile counter can be updated just after a test run of the new software when its health check is done. Just in case of successful health check can the counter updated to avoid the prevention of the revert. This might require a secondary restart of the device.

Tool support

There is a Python script, imgtool.py which is used to prepare the new image for upgrade (add header, sign the image, append TLV section). This script must be modified to get an additional command line argument which serves as security counter. The security counter must be included in the integrity protected part of the TLV section.

Copyright (c) 2019-2020, Arm Limited. All rights reserved.

HW crypto key integration in TF-M secure boot

Author
Tamas Ban

Organization
Arm Limited

Contact
Tamas Ban <tamas.ban@arm.com>

Abstract

PSA Trusted Boot and Firmware Update specification requires the support of at least one immutable root of trust public key (ROTPK) for firmware verification. This can be stored using a locked on-chip flash memory, a secure-element or on-chip OTP memory. It also beneficial to be able to provision these keys during the factory life-cycle of the device independently from any software components. The current key handling solution in TF-M secure boot does not supports this key provisioning process. MCUBoot requires compile time built-in public key(s) for image verification. This limitation does not fit well with a scenario with multiple vendors where multiple MCU software components might be deployed by different vendors in different points in the life-cycle of the device and they do not want to share the keys in-advance for embedding in the bootloader code. The goal of this document to propose a solution to decouple MCUBoot from public key(s) and enable the independent deployment of ROTPK.

Existing key handling solution

MCUBoot code contains a compile-time built-in key array which can store any number of key(s) for firmware verification: bl2/ext/mcuboot/keys.c. These public key(s) must be available when MCUBoot image is built. There is a script bl2/ext/mcuboot/scipt/imgtool.py which can generate a new key pair, and extract the public key part in the expected ASN.1 format and encode it as C structure. The script is also capable of signing the image with the private key. In order to identify and validate the corresponding public key during image verification the hash of the public key is appended to the image manifest area (TLV encoded metadata). During image verification the bootloader retrieves the hash of the public key from the manifest area and compare against the on-the-fly calculated hash value of the built-in public keys. An exact match identifies and validates the public key which must be used for image verification.

Current memory layout:

```
|--------------------|
|                    |
|     MCUBoot code     |
|                    |
|   Full public key    |
```

(continues on next page)

(continued from previous page)

```
|                     |
|---------------------|
|                     |
|       Image         |
|                     |
|---------------------|
|  Image Manifest(TLV) |
|                     |
|  Hash of public key  |
|---------------------|
|                     |
|   Rest of memory    |
|                     |
```

Requirements

- Multiple independent vendor scenario must be supported, when more than one ROTPK is provisioned to the device.
- The corresponding public key for image verification must be identifiable and verifiable.
- Key identity must be immutable and anchored to the device itself.
- Key(s) can be provisioned independently from bootloader.

Design proposal

HW key(s) might stored in OTP memory which is an expensive resource, so storing a large key (such as RSA) is inefficient. Therefore, it is assumed that only the hash of the ROTPK will be stored in the HW. If only the hash of the public key is stored in the HW then the whole public key must be transfered to the device, because it must be available during image verification. This transfer can be done in the same way as how the hash of the key is transfered to the device with the current solution. A new TLV type (TLV_KEY) can be introduced to carry the whole public key. The corresponding public key will be appended to the image itself in the manifest area. It has the drawback that the image will be bigger, but it is assumed that the additional cost of the bigger image (extra flash area + power for download) is less than the cost of the OTP area.

The verification flow:

- Look up the TLV_KEY field to get the public key.
- Calculates its hash(SHA256) value.
- Get the hash of ROTPK from the platform layer (stored in HW)
- Compare the two hash values, if they match then the key can be used to validate the image. In case of failure consider the images as unauthenticated.

Proposed memory layout:

```
|---------------------|
|                     |
|    MCUBoot code     |
|                     |
|   NO PUBLIC KEY     |
|                     |
|---------------------|
|                     |
|       Image         |
|                     |
|---------------------|
|  Image Manifest(TLV) |
|                     |
```

(continues on next page)

(continued from previous page)

```
|   Full public key    |
|----------------------|
|                      |
|                      |
|                      |
|   Rest of memory     |
|                      |
|                      |
|----------------------|
|   Immutable memory   |
|                      |
|  Hash of public key  |
|----------------------|
|                      |
```

Platform support

A new platform API used by the bootloader must be introduced to retrieve the image corresponding hash of ROTPK:

enum tfm_plat_err_t tfm_plat_get_rotpk_hash(uint8_t image_id, uint8_t *rotpk_hash, uint32_t *rotpk_hash_size);

The mapping between image identity and public key can be hard-code in the platform layer. This simplifies the validation of the public key, because no look-up would be needed. It is assumed that the memory area of the ROTPK hash is not directly accessible, therefore a buffer is allocated by the caller to store the hash there.

Compile time configurability

The solution must be compile time configurable in order to be able to switch between built-in key(s) and HW key(s) support, and also due to backwards compatibility.

Tooling

bl2/ext/mcuboot/scipt/imgtool.py will be modified to include the whole public key to the TLV area (instead of the hash).

Table to compare the current and proposed solution:

```
|---------|----------------------|------------------|--------------------|
|         |Key format in manifest |Key in MCUBoot code|     Key in HW      |
|---------|----------------------|------------------|--------------------|
|Proposed |    Full public key    |  No key embedded  | Hash of public key |
|---------|----------------------|------------------|--------------------|
|Current  |   Hash of public key  |  Full public key  |    No key in HW    |
|---------|----------------------|------------------|--------------------|
```

Copyright (c) 2019, Arm Limited. All rights reserved.

For secure devices it is security critical to enforce firmware authenticity to protect against execution of malicious software. This is implemented by building a trust chain where each step in the execution chain authenticates the next step before execution. The chain of trust is based on a “Root of Trust” which is implemented using asymmetric cryptography. The Root of Trust is a combination of an immutable bootloader and a public key (ROTPK).

Warning: In order to implement a proper chain of trust functionality, it is mandatory that the first stage bootloader and ROTPK is stored in an **immutable** way. To achieve this the bootloader code must be stored and executed from ROM or such part of flash memory which

supports write protection. ROTPK can be stored in a one-time-programmable (OTP) memory. If the SoC has a built-in BL1 (immutable) bootloader and the immutability of TF-M secure boot code is not guaranteed then TF-M secure boot code must be authenticated by BL1 bootloader before execution. If immutability of root of trust (first stage bootloader + ROTPK) is not ensured then there is a risk that the secure boot process could be bypassed, which could lead to arbitrary code execution on the device. Current TF-M secure boot code is intended to be a second stage bootloader, therefore it requires authentication before execution. If TF-M secure boot code is used as a first stage bootloader then it must be stored according to the above requirements.

Second stage bootloader in TF-M

By default, the MCUboot project from GitHub is used as the secure bootloader in TF-M. The repository is going to be automatically downloaded by CMake. The version downloaded can be controlled by the MCUBOOT_VERSION CMake variable. If you wish to use a locally downloaded copy, the CMake variable MCUBOOT_PATH can be set to its location. This document contains information about how MCUboot has been integrated to TF-M. For further information about MCUboot design please refer to the MCUBoot homepage.

Bootloader is started when CPU is released from reset. It runs in secure mode. It authenticates the firmware image by hash (SHA-256) and digital signature (RSA-3072) validation. Public key, that the checks happens against, can be built into the bootloader image or can be provisioned to the SoC during manufacturing. Metadata of the image is delivered together with the image itself in a header and trailer section. In case of successful authentication, bootloader passes execution to the secure image. Execution never returns to bootloader until next reset.

A default RSA key pair is stored in the repository, public key is in keys.c and private key is in root-RSA-3072.pem.

Danger: DO NOT use the default keys in a production code, they are exclusively for testing!

The private key must be stored in a safe place outside of the repository. imgtool.py (found in the scripts directory in the MCUBoot repository, or installed through the pip package manager) can be used to generate new key pairs.

The bootloader can handle the secure and non-secure images independently (multiple image boot) or together (single image boot). In case of multiple image boot they are signed independently with different keys and they can be updated separately. In case of single image boot the secure and non-secure image is handled as a single blob, therefore they must be contiguous in the device memory. In this case they are signed together and also they can be updated only together. In order to have the same artefacts at the end of the build regardless of how the images are handled (independently or together) the images are always concatenated. In case of single image boot they are concatenated first and then signed. In case of multiple image boot they are separately signed first and then concatenated. Preparation of payload is done by Python scripts: bl2/ext/mcuboot/scripts/. At the end of a successful build the signed TF-M payload can be found in: <build_dir>/bin/tfm_s_ns_signed.bin

Integration with TF-M

MCUBoot assumes a predefined memory layout which is described below (applicable for AN521). It is mandatory to define the primary slot and the secondary slot partitions, but their size and location can be changed:

```
- 0x0000_0000 - 0x0007_FFFF:    BL2 bootloader - MCUBoot
- 0x0008_0000 - 0x000F_FFFF:    Primary slot : Single binary blob:
                                Secure + Non-Secure image;
                                Primary memory partition
```

(continues on next page)

(continued from previous page)

```
  - 0x0008_0000 - 0x0008_03FF:  Common image header
  - 0x0008_0400 - 0x0008_xxxx:  Secure image
  - 0x0008_xxxx - 0x0010_03FF:  Padding (with 0xFF)
  - 0x0010_0400 - 0x0010_xxxx:  Non-secure image
  - 0x0010_xxxx - 0x0010_xxxx:  Hash value(SHA256), RSA signature and other
                                metadata of combined image

- 0x0018_0000 - 0x0027_FFFF:    Secondary slot : Secure + Non-Secure image;
                                Secondary memory partition, structured
                                identically to the primary slot
- 0x0028_0000 - 0x0037_FFFF:    Scratch area, only used during image
                                swapping
```

Multiple image boot requires a slightly different layout:

```
- 0x0000_0000 - 0x0007_FFFF:    BL2 bootloader - MCUBoot
- 0x0008_0000 - 0x000F_FFFF:    Primary slot : Secure image
  - 0x0008_0000 - 0x0008_03FF:  Secure image header
  - 0x0008_0400 - 0x000x_xxxx:  Secure image
  - 0x000x_xxxx - 0x000x_xxxx:  Hash value(SHA256), RSA signature and other
                                metadata of secure image

- 0x0010_0000 - 0x0017_FFFF:    Primary slot : Non-secure image
  - 0x0010_0000 - 0x0010_03FF:  Non-secure image header
  - 0x0010_0400 - 0x001x_xxxx:  Non-secure image
  - 0x001x_xxxx - 0x001x_xxxx:  Hash value(SHA256), RSA signature and other
                                metadata of non-secure image

- 0x0018_0000 - 0x001F_FFFF:    Secondary slot : Secure image
- 0x0020_0000 - 0x0027_FFFF:    Secondary slot : Non-secure image

- 0x0028_0000 - 0x002F_FFFF:    Scratch area, only used during image
                                swapping, used for secure and non-secure
                                image as well
```

Firmware upgrade operation

MCUBoot handles only the firmware authenticity check after start-up and the firmware switch part of the firmware update process. Downloading the new version of the firmware is out-of-scope for MCUBoot. MCUBoot supports three different ways to switch to the new firmware and it is assumed that firmware images are executed-in-place (XIP). The default behaviour is the overwrite-based image upgrade. In this case the active firmware is always executed from the primary slot and the secondary slot is a staging area for new images. Before executing the new firmware image, the content of the primary slot must be overwritten with the content of the secondary slot (the new firmware image). The second option is the image swapping strategy when the content of the two memory slots must be physically swapped. This needs the scratch area to be defined in the memory layout. The third option is the direct execute-in-place version, which eliminates the complexity of image swapping and its administration. Active image can be executed from either memory slot, but new firmware must be linked to the address space of the proper (currently inactive) memory slot.

Overwrite operation

Active image is stored in the primary slot, and this image is started always by the bootloader. Therefore images must be linked to the primary slot. If the bootloader finds a valid image in the secondary slot, which is marked for upgrade, then the content of the primary slot will be simply overwritten with the content of the secondary slot, before starting the new image from the primary slot. After the content of the primary slot has been successfully overwritten, the header and trailer of the new image in the secondary slot is erased to prevent the triggering

of another unnecessary image upgrade after a restart. The overwrite operation is fail-safe and resistant to power-cut failures. For more details please refer to the MCUBoot documentation.

Swapping operation

This operation can be set with the MCUBOOT_UPGRADE_STRATEGY compile time switch (see *Build time configuration*). With swapping image upgrade strategy the active image is also stored in the primary slot and it will always be started by the bootloader. If the bootloader finds a valid image in the secondary slot, which is marked for upgrade, then contents of the primary slot and the secondary slot will be swapped, before starting the new image from the primary slot. Scratch area is used as a temporary storage place during image swapping. Update mark from the secondary slot is removed when the swapping is successful. The boot loader can revert the swapping as a fall-back mechanism to recover the previous working firmware version after a faulty update. The swap operation is fail-safe and resistant to power-cut failures. For more details please refer to the MCUBoot documentation.

Note: After a successful image upgrade, user can mark the image as "OK" at runtime by explicitly calling psa_fwu_accept. When this happens, the swap is made "permanent" and MCUBoot will then still choose to run it during the next boot.

TF-M does not set the image_ok flag, because it is user's duty to determine whether the image is acceptable. Therefore the bootloader will always perform a "revert" (swap the images back) during the next boot.

Direct execute-in-place operation

This operation can be set with the MCUBOOT_UPGRADE_STRATEGY compile time switch (see *Build time configuration*). When enabling direct-xip operation then the active image flag is moved between slots during firmware upgrade. If firmware is executed-in-place (XIP), then two firmware images must be generated. One of them is linked to be executed from the primary slot memory region and the other from the secondary slot. The firmware upgrade client, which downloads the new image, must be aware, which slot hosts the active firmware and which acts as a staging area and it is responsible for downloading the proper firmware image. At boot time MCUBoot inspects the version number in the image header and passes execution to the newer firmware version. New image must be marked for upgrade which is automatically done by Python scripts at compile time. Image verification is done the same way in all operational modes. If new image fails during authentication then MCUBoot erases the memory slot and starts the other image, after successful authentication.

To select which slot the image is to be executed from, set MCUBOOT_EXECUTION_SLOT to the desired index. It is suggested that you create two build directories when building images using this mode, as intermediate dependencies cannot be reused due to changes in the flash layout.

Note: Only single image boot is supported with direct-xip upgrade mode.

RAM Loading firmware upgrade

Musca-S supports an image upgrade mode that is separate to the other (overwrite, swapping and direxT-xip) modes. This is the RAM load mode (please refer to the table below). Like the direct-xip mode, this selects the newest image by reading the image version numbers in the image headers, but instead of executing it in place, the newest image is copied to RAM for execution. The load address, the location in RAM where the image is copied to, is stored in the image header.

Summary of different modes for image upgrade

Different implementations of the image upgrade operation (whether through overwriting, swapping, direct-xip or loading into RAM and executing from there) are supported by the platforms. The table below shows which of these modes are supported by which platforms:

	Without BL2[1]	With BL2[2]			
	XIP	XIP Over-write[3]	XIP Swap[4]	XIP direct-xip[5]	Not XIP RAM load[6]
AN521	Yes	Yes	Yes	Yes	No
AN519	Yes	Yes	Yes	Yes	No
FVP_SSE300_MPS3	No	Yes	Yes	Yes	No
Corstone-310 FVP	Yes	Yes	Yes	Yes	No
LPC55S69	Yes	Yes	No	Yes	No
Musca-B1	Yes	Yes	Yes	Yes	No
Musca-S1	Yes	Yes	Yes	Yes	No
AN524	Yes	No	No	Yes	No
AN547	No	Yes	Yes	Yes	No
AN552	No	Yes	Yes	Yes	No
PSoC64	Yes	No	No	No	No
STM_DISCO_L562QE	No	Yes	No	No	No
STM_NUCLEO_L552ZE_Q	No	Yes	No	No	No
nRF9160 DK	Yes	Yes	No	No	No
nRF5340 DK	Yes	Yes	No	No	No
BL5340 DVK	Yes	Yes	Yes	No	No
RSE	No	No	No	No	Yes

Multiple image boot

It is possible to update the firmware images independently to support the scenario when secure and non-secure images are provided by different vendors. Multiple image boot is supported only together with the overwrite and swap firmware upgrade modes.

It is possible to describe the dependencies of the images on each other in order to avoid a faulty upgrade when incompatible versions would be installed. These dependencies are part of the image manifest area. The dependencies are composed from two parts:

- **Image identifier:** The number of the image which the current image (whose manifest area contains the dependency entry) depends on. The image identifier starts from 0.
- **Minimum version:** The minimum version of other image must be present on the device by the end of the upgrade (both images might be updated at the same time).

Dependencies can be added to the images at compile time with the following compile time switches:

- MCUBOOT_S_IMAGE_MIN_VER It is added to the non-secure image and specifies the minimum required version of the secure image.
- MCUBOOT_NS_IMAGE_MIN_VER It is added to the secure image and specifies the minimum required version of the non-secure image.

[1] To disable BL2, please set the BL2 cmake option to OFF
[2] BL2 is enabled by default
[3] The image executes in-place (XIP) and is in Overwrite mode for image update by default
[4] To enable XIP Swap mode, assign the “SWAP_USING_SCRATCH” or “SWAP_USING_MOVE” string to the MCUBOOT_UPGRADE_STRATEGY configuration variable in the build configuration file, or include this macro definition in the command line
[5] To enable direct-xip, assign the “DIRECT_XIP” string to the MCUBOOT_UPGRADE_STRATEGY configuration variable in the build configuration file, or include this macro definition in the command line
[6] To enable RAM load, assign the “RAM_LOAD” string to the MCUBOOT_UPGRADE_STRATEGY configuration variable in the build configuration file, or include this macro definition in the command line

Example of how to provide the secure image minimum version:

```
cmake -DTFM_PLATFORM=arm/musca_b1 -DMCUBOOT_S_IMAGE_MIN_VER=1.2.3+4 ..
```

Signature algorithms

MbedTLS library is used to sign the images. The list of supported signing algorithms:

- *RSA-2048*
- *RSA-3072*: default

Example keys stored in:

- root-RSA-2048.pem : Used to sign single image (S+NS) or secure image in case of multiple image boot
- root-RSA-2048_1.pem : Used to sign non-secure image in case of multiple image boot
- root-RSA-3072.pem : Used to sign single image (S+NS) or secure image in case of multiple image boot
- root-RSA-3072_1.pem : Used to sign non-secure image in case of multiple image boot

Build time configuration

MCUBoot related compile time switches can be set by cmake variables.

- **BL2 (default: True):**
  - **True:** TF-M built together with bootloader. MCUBoot is executed after reset and it authenticates TF-M and starts secure code.
  - **False:** TF-M built without bootloader. Secure image linked to the beginning of the device memory and executed after reset. If it is false then using any of the further compile time switches is invalid.
- **MCUBOOT_UPGRADE_STRATEGY (default: "OVERWRITE_ONLY"):**
  - **"OVERWRITE_ONLY":** Default firmware upgrade operation with overwrite.
  - **"SWAP_USING_SCRATCH":** Activate swapping firmware upgrade operation with a scratch area in flash
  - **"SWAP_USING_MOVE":** Activate swapping firmware upgrade operation without a scratch area in flash
  - **"DIRECT_XIP":** Activate direct execute-in-place firmware upgrade operation.
  - **"RAM_LOAD":** Activate RAM loading firmware upgrade operation, where the latest image is copied to RAM and runs from there instead of being executed in-place.
- **MCUBOOT_SIGNATURE_TYPE (default: RSA-3072):**
  - **RSA-2048:** Image is signed with RSA algorithm and signed with 2048 bit key.
  - **RSA-3072:** Image is signed with RSA algorithm and signed with 3072 bit key.
  - **EC-P256:** Image is signed with ECDSA P-256 algorithm.
  - **EC-P384:** Image is signed with ECDSA P-384 algorithm.
- **MCUBOOT_IMAGE_NUMBER (default: 2):**
  - **1:** Single image boot, secure and non-secure images are signed and updated together.
  - **2:** Multiple image boot, secure and non-secure images are signed and updatable independently.
- **MCUBOOT_HW_KEY (default: True):**

  - **True:** The hash of public key is provisioned to the SoC and the image manifest contains the whole public key (imgtool uses --public_key_format=full). MCUBoot validates the key before using it for firmware authentication, it calculates the hash of public key from the manifest and compare against the retrieved key-hash from the hardware. This way MCUBoot is independent from the public key(s). Key(s) can be provisioned any time and by different parties.
  - **False:** The whole public key is embedded to the bootloader code and the image manifest contains only the hash of the public key (imgtool uses --public_key_format=hash). MCUBoot validates the key before using it for firmware authentication, it calculates the hash of built-in public key and compare against the retrieved key-hash from the image manifest. After this the bootloader can verify that the image was signed with a private key that corresponds to the retrieved key-hash (it can have more public keys embedded in and it may have to look for the matching one). All the public key(s) must be known at MCUBoot build time.

- **MCUBOOT_BUILTIN_KEY (default: False):**
  - **True:** When enabled, the entire public key used for signature verification must be provisioned to the target device. In this case, neither the code nor the image metadata needs to contain any public key data. During image validation only a key ID is passed to the verifier function for the required key to be selected. The key handling is entirely the responsibility of the underlying crypto library and the details of the key handling mechanism are abstracted away from the boot code.

- **MCUBOOT_LOG_LEVEL:**
  Can be used to configure the level of logging in MCUBoot. The possible values are the following:
  - **OFF**
  - **ERROR**
  - **WARNING**
  - **INFO**
  - **DEBUG**

  The logging in MCUBoot can be disabled and thus the code size can be reduced by setting it to OFF. Its value depends on the build type. If the build type is Debug then default value is INFO. In case of different kinds of Release builds the default value is OFF. The default value can be overridden through the command line or in the CMake GUI regardless of the build type.

- **MCUBOOT_ENC_IMAGES (default: False):**
  - **True:** Adds encrypted image support in the source and encrypts the resulting image using the enc-rsa2048-pub.pem key found in the MCUBoot repository.
  - **False:** Doesn't add encrypted image support and doesn't encrypt the image.

Note: The decryption takes place during the upgrade process, when the images are being moved between the slots. This means that boards that don't already have an image on them with MCUBoot that has been compiled with MCUBOOT_ENCRYPT_RSA enabled need special treatment. In order to load an encrypted image to such boards, an upgrade needs to be executed. This can be done by using MCUBoot, putting an image in the secondary image area, and setting MCUBOOT_ENCRYPT_RSA to ON. When using the OVERWRITE_ONLY upgrade strategy, this is enough. When using SWAP_USING_SCRATCH or SWAP_USING_MOVE, an image is needed in the primary image area as well, to trigger the update.

> **Danger:** DO NOT use the enc-rsa2048-pub.pem key in production code, it is exclusively for testing!

Image versioning

An image version number is written to its header by one of the Python scripts, and this number is used by the bootloader when the direct execute-in-place or the RAM loading mode is enabled. It is also used in case of multiple image boot when the bootloader checks the image dependencies if any have been added to the images.

The version number of the image (single image boot) can manually be passed in through the command line in the cmake configuration step:

```
cmake -DTFM_PLATFORM=arm/musca_b1 -DIMAGE_VERSION_S=1.2.3+4 ..
```

Alternatively, the version number can be less specific (e.g 1, 1.2, or 1.2.3), where the missing numbers are automatically set to zero. The image version number argument is optional, and if it is left out, then the version numbers of the image(s) being built in the same directory will automatically change. In this case, the last component (the build number) automatically increments from the previous one: 0.0.0+1 -> 0.0.0+2, for as many times as the build is re-ran, **until a number is explicitly provided**. If automatic versioning is in place and then an image version number is provided for the first time, the new number will take precedence and be used instead. All subsequent image versions are then set to the last number that has been specified, and the build number would stop incrementing. Any new version numbers that are provided will overwrite the previous one: 0.0.0+1 -> 0.0.0+2. Note: To re-apply automatic image versioning, please start a clean build without specifying the image version number at all. In case of multiple image boot there are separate compile time switches for both images to provide their version: IMAGE_VERSION_S and IMAGE_VERSION_NS. These must be used instead of IMAGE_VERSION_S.

Security counter

Each signed image contains a security counter in its manifest. It is used by the bootloader and its aim is to have an independent (from the image version) counter to ensure rollback protection by comparing the new image's security counter against the original (currently active) image's security counter during the image upgrade process. It is added to the manifest (to the TLV area that is appended to the end of the image) by one of the Python scripts when signing the image. The value of the security counter is security critical data and it is in the integrity protected part of the image. The last valid security counter should always be stored in a non-volatile and trusted component of the device and its value should always be increased if a security flaw was fixed in the current image version. The value of the security counter (single image boot) can be specified at build time in the cmake configuration step:

```
cmake -DTFM_PLATFORM=arm/musca_b1 -DSECURITY_COUNTER_S=42 ../
```

The security counter can be independent from the image version, but not necessarily. Alternatively, if it is not specified at build time with the SECURITY_COUNTER option the Python script will automatically generate it from the image version number (not including the build number) and this value will be added to the signed image. In case of multiple image boot there are separate compile time switches for both images to provide their security counter value: SECURITY_COUNTER_S and SECURITY_COUNTER_NS. These must be used instead of SECURITY_COUNTER_S. If these are not defined then the security counter values will be derived from the corresponding image version similar to the single image boot.

Signing the images manually

Normally the build system handles the signing (computing hash over the image and security critical manifest data and then signing the hash) of the firmware images. However, the images

also can be signed manually by using the imgtool Python program which is located in the MCUboot repository in the scripts folder or can be installed with the pip package manager. Issue the python3 imgtool.py sign --help command in the directory for more information about the mandatory and optional arguments. The tool takes an image in binary or Intel Hex format and adds a header and trailer that MCUBoot is expecting. In case of single image boot after a successful build the tfm_s_ns.bin build artifact (contains the concatenated secure and non-secure images) must be passed to the script and in case of multiple image boot the tfm_s.bin and tfm_ns.bin binaries can be passed to prepare the signed images.

Signing the secure image manually in case of multiple image boot

```
python3 bl2/ext/mcuboot/scripts/imgtool.py sign \
    --layout <build_dir>/bl2/ext/mcuboot/CMakeFiles/signing_layout_s.dir/signing_layout_s.c.obj \
    -k <tfm_dir>/bl2/ext/mcuboot/root-RSA-3072.pem \
    --public-key-format full \
    --align 1 \
    -v 1.2.3+4 \
    -d "(1,1.2.3+0)" \
    -s 42 \
    -H 0x400 \
    <build_dir>/bin/tfm_s.bin \
    <build_dir>/bin/tfm_s_signed.bin
```

Testing firmware upgrade

As downloading the new firmware image is out of scope for MCUBoot, the update process is started from a state where the original and the new image are already programmed to the appropriate memory slots. To generate the original and a new firmware package, TF-M is built twice with different build configurations.

Overwriting firmware upgrade

Run TF-M build twice with MCUBOOT_IMAGE_NUMBER set to "1" in both cases (single image boot), but with two different build configurations: default and regression. Save the artifacts between builds, because second run can overwrite original binaries. Download default build to the primary slot and regression build to the secondary slot.

Executing firmware upgrade on FVP_MPS2_AEMv8M

```
<ARM_DS_PATH>/sw/models/bin/FVP_MPS2_AEMv8M  \
--parameter fvp_mps2.platform_type=2 \
--parameter cpu0.baseline=0 \
--parameter cpu0.INITVTOR_S=0x10000000 \
--parameter cpu0.semihosting-enable=0 \
--parameter fvp_mps2.DISABLE_GATING=0 \
--parameter fvp_mps2.telnetterminal0.start_telnet=1 \
--parameter fvp_mps2.telnetterminal1.start_telnet=0 \
--parameter fvp_mps2.telnetterminal2.start_telnet=0 \
--parameter fvp_mps2.telnetterminal0.quiet=0 \
--parameter fvp_mps2.telnetterminal1.quiet=1 \
--parameter fvp_mps2.telnetterminal2.quiet=1 \
--application cpu0=<build_dir>/bin/bl2.axf \
--data cpu0=<default_build_dir>/bin/tfm_s_ns_signed.bin@0x10080000 \
--data cpu0=<regresssion_build_dir>/bin/tfm_s_ns_signed.bin@0x10180000
```

Executing firmware upgrade on SSE 200 FPGA on MPS2 board

```
TITLE: Versatile Express Images Configuration File
[IMAGES]
TOTALIMAGES: 3                ;Number of Images (Max: 32)
IMAGE0ADDRESS: 0x00000000
IMAGE0FILE: \Software\bl2.axf      ; BL2 bootloader
IMAGE1ADDRESS: 0x10080000
IMAGE1FILE: \Software\tfm_sig1.bin ; TF-M default test binary blob
IMAGE2ADDRESS: 0x10180000
IMAGE2FILE: \Software\tfm_sig2.bin ; TF-M regression test binary blob
```

The following message will be shown in case of successful firmware upgrade:

```
[INF] Starting bootloader
[INF] Swap type: test
[INF] Image upgrade secondary slot -> primary slot
[INF] Erasing the primary slot
[INF] Copying the secondary slot to the primary slot: 0x100000 bytes
[INF] Bootloader chainload address offset: 0x80000
[INF] Jumping to the first image slot
[Sec Thread] Secure image initializing!

#### Execute test suites for the Secure area ####
Running Test Suite PSA protected storage S interface tests (TFM_PS_TEST_2XXX)...
...
```

To update the secure and non-secure images separately (multiple image boot), set the MCUBOOT_IMAGE_NUMBER switch to "2" (this is the default configuration value) and follow the same instructions as in case of single image boot.

Executing multiple firmware upgrades on SSE 200 FPGA on MPS2 board

```
TITLE: Versatile Express Images Configuration File
[IMAGES]
TOTALIMAGES: 4                ;Number of Images (Max: 32)
IMAGE0ADDRESS: 0x00000000
IMAGE0FILE: \Software\bl2.axf      ; BL2 bootloader
IMAGE1ADDRESS: 0x10080000
IMAGE1FILE: \Software\tfm_sign.bin ; TF-M default test binary blob
IMAGE2ADDRESS: 0x10180000
IMAGE2FILE: \Software\tfm_ss1.bin  ; TF-M regression test secure (signed) image
IMAGE3ADDRESS: 0x10200000
IMAGE3FILE: \Software\tfm_nss1.bin ; TF-M regression test non-secure (signed) image
```

Note that both the concatenated binary blob (the images are signed separately and then concatenated) and the separate signed images can be downloaded to the device because on this platform (AN521) both the primary slots and the secondary slots are contiguous areas in the Flash (see *Integration with TF-M*). The following message will be shown in case of successful firmware upgrades:

```
[INF] Starting bootloader
[INF] Swap type: test
[INF] Swap type: test
[INF] Image upgrade secondary slot -> primary slot
[INF] Erasing the primary slot
[INF] Copying the secondary slot to the primary slot: 0x80000 bytes
[INF] Image upgrade secondary slot -> primary slot
[INF] Erasing the primary slot
[INF] Copying the secondary slot to the primary slot: 0x80000 bytes
[INF] Bootloader chainload address offset: 0x80000
[INF] Jumping to the first image slot
[Sec Thread] Secure image initializing!
```

(continues on next page)

(continued from previous page)

```
TFM level is: 1
[Sec Thread] Jumping to non-secure code...

#### Execute test suites for the Secure area ####
Running Test Suite PSA protected storage S interface tests (TFM_PS_TEST_2XXX)...
...
```

Swapping firmware upgrade

Follow the same instructions and platform related configurations as in case of overwriting build including these changes:

- Set the MCUBOOT_UPGRADE_STRATEGY compile time switch to “SWAP” before build.
- Set the MCUBOOT_IMAGE_NUMBER compile time switch to “1” (single image boot) or “2” (multiple image boot) before build.

During single image boot the following message will be shown in case of successful firmware upgrade, Swap type: test indicates that images were swapped:

```
[INF] Starting bootloader
[INF] Image 0: magic= good, copy_done=0x3, image_ok=0x3
[INF] Scratch: magic=  bad, copy_done=0x0, image_ok=0x2
[INF] Boot source: primary slot
[INF] Swap type: test
[INF] Bootloader chainload address offset: 0x80000
[INF] Jumping to the first image slot
[Sec Thread] Secure image initializing!

#### Execute test suites for the Secure area ####
Running Test Suite PSA protected storage S interface tests (TFM_PS_TEST_2XXX)...
...
```

Direct execute-in-place firmware upgrade

Follow the same instructions and platform related configurations as in case of overwriting build including these changes:

- Set the MCUBOOT_UPGRADE_STRATEGY compile time switch to “DIRECT_XIP” before build.
- set MCUBOOT_EXECUTION_SLOT to 1 in the regression build dir.
- Make sure the image version number was increased between the two build runs either by specifying it manually or by checking in the build log that it was incremented automatically.

Executing firmware upgrade on FVP_MPS2_AEMv8M

```
<ARM_DS_PATH>/sw/models/bin/FVP_MPS2_AEMv8M  \
--parameter fvp_mps2.platform_type=2 \
--parameter cpu0.baseline=0 \
--parameter cpu0.INITVTOR_S=0x10000000 \
--parameter cpu0.semihosting-enable=0 \
--parameter fvp_mps2.DISABLE_GATING=0 \
--parameter fvp_mps2.telnetterminal0.start_telnet=1 \
--parameter fvp_mps2.telnetterminal1.start_telnet=0 \
--parameter fvp_mps2.telnetterminal2.start_telnet=0 \
--parameter fvp_mps2.telnetterminal0.quiet=0 \
--parameter fvp_mps2.telnetterminal1.quiet=1 \
--parameter fvp_mps2.telnetterminal2.quiet=1 \
--application cpu0=<build_dir>/bin/bl2.axf \
```

(continues on next page)

(continued from previous page)

```
--data cpu0=<default_build_dir>/bin/tfm_s_ns_signed.bin@0x10080000 \
--data cpu0=<regresssion_build_dir>/bin/tfm_s_ns_signed.bin@0x10180000
```

Executing firmware upgrade on SSE 200 FPGA on MPS2 board

```
TITLE: Versatile Express Images Configuration File
[IMAGES]
TOTALIMAGES: 3                  ;Number of Images (Max: 32)
IMAGE0ADDRESS: 0x00000000
IMAGE0FILE: \Software\bl2.axf      ; BL2 bootloader
IMAGE1ADDRESS: 0x10080000
IMAGE1FILE: \Software\tfm_sign.bin ; TF-M default test binary blob
IMAGE2ADDRESS: 0x10180000
IMAGE2FILE: \Software\tfm_sig1.bin ; TF-M regression test binary blob
```

Executing firmware upgrade on Musca-B1 and Musca-S1 boards

After the two images have been built, they can be concatenated to create the combined image using srec_cat:

- Linux:

```
srec_cat bin/bl2.bin -Binary -offset 0xA000000 tfm_sign.bin -Binary -offset 0xA020000 tfm_sign_1.bin␣
↪-Binary -offset 0xA100000 -o tfm.hex -Intel
```

- Windows:

```
srec_cat.exe bin\bl2.bin -Binary -offset 0xA000000 tfm_sign.bin -Binary -offset 0xA020000 tfm_sign_1.
↪bin -Binary -offset 0xA100000 -o tfm.hex -Intel
```

The following message will be shown in case of successful firmware upgrade, notice that image with higher version number (version=1.2.3.5) is executed:

```
[INF] Starting bootloader
[INF] Image 0: version=1.2.3.4, magic= good, image_ok=0x3
[INF] Image 1: version=1.2.3.5, magic= good, image_ok=0x3
[INF] Booting image from the secondary slot
[INF] Bootloader chainload address offset: 0xa0000
[INF] Jumping to the first image slot
[Sec Thread] Secure image initializing!

#### Execute test suites for the Secure area ####
Running Test Suite PSA protected storage S interface tests (TFM_PS_TEST_2XXX)...
...
```

Executing firmware upgrade on CoreLink SSE-200 Subsystem for MPS3 (AN524)

```
TITLE: Arm MPS3 FPGA prototyping board Images Configuration File

[IMAGES]
TOTALIMAGES: 3                  ;Number of Images (Max: 32)

IMAGE0UPDATE: AUTO              ;Image Update:NONE/AUTO/FORCE
IMAGE0ADDRESS: 0x00000000
IMAGE0FILE: \SOFTWARE\bl2.bin      ;BL2 bootloader
IMAGE1UPDATE: AUTO
IMAGE1ADDRESS: 0x00040000
IMAGE1FILE: \SOFTWARE\tfm_sig0.bin ;TF-M example application binary blob
IMAGE2UPDATE: AUTO
```

(continues on next page)

(continued from previous page)

```
IMAGE2ADDRESS: 0x000C0000
IMAGE2FILE: \SOFTWARE\tfm_sig1.bin ;TF-M regression test binary blob
```

RAM loading firmware upgrade

To enable RAM loading, please set MCUBOOT_UPGRADE_STRATEGY to "RAM_LOAD" (either in the configuration file or through the command line), and then specify a destination load address in RAM where the image can be copied to and executed from. The S_IMAGE_LOAD_ADDRESS macro must be specified in the target dependent files, and if multiple image boot is enabled then NS_IMAGE_LOAD_ADDRESS must also be defined. For example with Musca-S, its flash_layout.h file in the platform folder should include #define S_IMAGE_LOAD_ADDRESS #0xA0020000

Executing firmware upgrade on Musca-S board

After two images have been built, they can be concatenated to create the combined image using srec_cat:

- Linux:

```
srec_cat bin/bl2.bin -Binary -offset 0xA000000 tfm_sign_old.bin -Binary -offset 0xA020000 tfm_sign_
↪new.bin -Binary -offset 0xA100000 -o tfm.hex -Intel
```

- Windows:

```
srec_cat.exe bin\bl2.bin -Binary -offset 0xA000000 tfm_sign_old.bin -Binary -offset 0xA020000 tfm_
↪sign_new.bin -Binary -offset 0xA100000 -o tfm.hex -Intel
```

The following message will be shown in case of successful firmware upgrade when, RAM loading is enabled, notice that image with higher version number (version=0.0.0.2) is executed:

```
[INF] Starting bootloader
[INF] Image 0: version=0.0.0.1, magic= good, image_ok=0x3
[INF] Image 1: version=0.0.0.2, magic= good, image_ok=0x3
[INF] Image has been copied from the secondary slot in flash to SRAM address 0xA0020000
[INF] Booting image from SRAM at address 0xA0020000
[INF] Bootloader chainload address offset: 0x20000
[INF] Jumping to the first image slot
[Sec Thread] Secure image initializing!
```

Integration with Firmware Update service

The shim layer of the Firmware Update partition calls the APIs in bootutil_misc.c to control the image status.

- Call boot_set_pending_multi() to make the image as a candidate image for booting.
- Call boot_set_confirmed_multi() to make the image as a permanent image.

Note: Currently, in direct-xip mode and ram-load mode, TF-M cannot get the information of which slot contains the running image from the bootloader. So the Firmware Update partition cannot decide where to write the new image. As a result, the firmware update service is not supported in direct-xip mode and ram-load mode.

Copyright (c) 2018-2024, Arm Limited. All rights reserved.

Copyright (c) 2023, Arm Limited. All rights reserved.

Dual-CPU

Booting a Dual-Core System

Author
Chris Brand

Organization
Cypress Semiconductor Corporation

Contact
chris.brand@cypress.com

System Architecture

There are many possibly ways to design a dual core system. Some important considerations from a boot perspective are:

- Which core has access to which areas of Flash?
  - It is possible that the secure core has no access to the Flash from which the non-secure core will boot, in which case the non-secure core will presumably have a separate root of trust and perform its own integrity checks on boot.
- How does the non-secure core behave on power-up? Is it held in reset, does it jump to a set address, ...?
- What are the performance characteristics of the two core?
  - There could be a great disparity in performance

TF-M Twin Core Booting

In an effort to make the problem manageable, as well as to provide a system with good performance, that is flexible enough to work for a variety of dual core systems, the following design decisions have been made:

- TF-M will (for now) only support systems where the secure core has full access to the Flash that the non-secure core will boot from
  - This keeps the boot flow as close as possible to the single core design, with the secure core responsible for maintaining the chain of trust for the entire system, and for upgrade of the entire system
- The secure code will make a platform-specific call immediately after setting up hardware protection to (potentially) start the non-secure core running
  - This is the earliest point at which it is safe to allow the non-secure code to start running, so starting it here ensures system integrity while also giving the non-secure code the maximum amount of time to perform its initialization
  - Note that this is after the bootloader has validated the non-secure image, which is the other key part to maintain security
  - This also means that only tfm_s and tfm_ns have to change, and not mcuboot
- Both the secure and non-secure code will make platform-specific calls to establish a synchronization point. This will be after both sides have done any initialization that is required, including setting up inter-core communications. On a single core system, this would be the point at which the secure code jumps to the non-secure code, and at the very start of the non-secure code.

- After completing initialization on the secure core (at the point where on a single core system, it would jump to the non-secure code), the main thread on the secure core will be allowed to die
  - The scheduler has been started at this point, and an idle thread exists. Any additional work that is only required in the dual core case will be interrupt-driven.
  - All work related to the non-secure core will take place from a ns_agent_mailbox partition, which will establish communication with the non-secure core and then act on its behalf
- Because both cores may be booting in parallel, executing different initialization code, at different speeds, the design must be resilient if either core attempts to communicate with the other before the latter is ready. For example, the client (non-secure) side of the IPC mechanism must be able to handle the situation where it has to wait for the server (secure) side to finish setting up the IPC mechanism.
  - This relates to the synchronization calls mentioned above. It means that those calls cannot utilise the IPC mechanism, but must instead use some platform-specific mechanism to establish this synchronization. This could be as simple as setting aside a small area of shared memory and having both sides set a "ready" flag, but may well also involve the use of interrupts.
  - This also means that the synchronization call must take place after the IPC mechanism has been set up but before any attempt (by either side) to use it.

API Additions

Three new HAL functions are required:

```
void tfm_hal_boot_ns_cpu(uintptr_t start_addr);
```

- Called on the secure core from ns_agent_mailbox partition when it first runs (after protections have been configured).
- Performs the necessary actions to start the non-secure core running the code at the specified address.

```
void tfm_hal_wait_for_ns_cpu_ready(void);
```

- Called on the secure core from ns_agent_mailbox partition after making the above call.
- Flags that the secure core has completed its initialization, including setting up the IPC mechanism.
- Waits, if necessary, for the non-secure core to flag that it has completed its initialisation

```
void tfm_ns_wait_for_s_cpu_ready(void);
```

- Called on the non-secure core from main() after the dual-core-specific initialization (on a single core system, this would be the start of the non-secure code), before the first use of the IPC mechanism.
- Flags that the non-secure side has completed its initialization.
- Waits, if necessary, for the secure core to flag that it has completed its initialization.

For all three, an empty implementation will be provided with a weak symbol so that platforms only have to provide the new functions if they are required.

Copyright (c) 2019-2022 Cypress Semiconductor Corporation. All rights reserved. Copyright (c) 2021, Arm Limited. All rights reserved.

Communication Prototype Between NSPE And SPE In Dual Core System

Author
David Hu

Organization
Arm Limited

Contact
david.hu@arm.com

Introduction

This document proposes a generic prototype of the communication between NSPE (Non-secure Processing Environment) and SPE (Secure Processing Environment) in TF-M on a dual core system.

The dual core system has the following requirements

- NSPE and SPE are properly isolated and protected following PSA
- An Arm M-profile core locates in SPE and acts as the Secure core
- An Inter-Processor Communication hardware module in system for communication between NSPE core and SPE core
- TF-M runs on the Secure core with platform specific drivers support.

Scope

This design document focuses on the dual core communication design inside TF-M. Some changes to TF-M core/Secure Partition Manager (SPM) are listed to support the dual core communication. This document only discusses the implementation in TF-M Inter-Process Communication (IPC) model. The TF-M non-secure interface library depends on mailbox and NS RTOS implementation. The related changes to TF-M non-secure interface library are not discussed in detail in this document.

Some requirements to mailbox functionalities are defined in this document. The detailed mailbox design or implementations is not specified in this document. Please refer to mailbox dedicated document[1].

Organization of the document

- *Overall workflow in dual core communication* provides an overall workflow of dual core communication between NSPE and SPE.
- *Requirements on mailbox communication* lists general requirements of mailbox, from the perspective of dual core communication.
- *PSA client call handling flow in TF-M* discusses about the detailed sequence and key modules of handling PSA client call in TF-M.
- *Summary of changes to TF-M core/SPM* summarizes the potential changes to TF-M core/SPM to support dual core communication.

Overall workflow in dual core communication

The overall workflow in dual-core scenario can be described as follows

1. Non-secure application calls TF-M non-secure interface library to request Secure service. The TF-M non-secure interface library translates the Secure service into PSA Client calls.
2. TF-M non-secure interface library notifies TF-M of the PSA client call request, via mailbox. Proper generic mailbox APIs in HAL must be defined so that TF-M non-secure interface library can co-work with diverse platform specific Inter-Processor Communication implementations.

[1] *Mailbox Design in TF-M on Dual-core System*

3. Inter-Processor Communication interrupt handler and mailbox handling in TF-M deal with the inbound mailbox event(s) and deliver the PSA client call request to TF-M SPM.
4. TF-M SPM processes the PSA client call request. The PSA client call is eventually handled in target Secure Partition or corresponding handler.
5. After the PSA Client call is completed, the return value is returned to NSPE via mailbox.
6. TF-M non-secure interface library fetches return value from mailbox.
7. The return value is returned to non-secure application.

The interfaces between NSPE app and TF-M NSPE interface library are unchanged so the underlying platform specific details are transparent to NSPE application.

Step 3 ~ step 5 are covered in *PSA client call handling flow in TF-M* in detail.

Requirements on mailbox communication

The communication between NSPE and SPE relies on mailbox communication implementation. The mailbox functionalities are eventually implemented by platform specific Inter-Processor Communication drivers. This section lists some general requirements on mailbox communication between NSPE and SPE.

Data transferred between NPSE and SPE

A mailbox message shall contain the information and parameters of a PSA client call. After SPE is notified by a mailbox event, SPE fetches the parameters from NSPE for PSA Client call processing. The mailbox design document[Page 1342, 1] defines the structure of the mailbox message.

The information and parameters of PSA Client call in the mailbox message include

- PSA Client API
- Parameters required in PSA Client call. The parameters can include the following, according to PSA client call type
  - Service ID (SID)
  - Handle
  - Request type
  - Input vectors and the lengths
  - Output vectors and the lengths
  - Requested version of secure service
- NSPE Client ID. Optional. The NSPE Client ID is required when NSPE RTOS enforces non-secure task isolation.

The mailbox implementation may define additional members in mailbox message to accomplish mailbox communication between NSPE and SPE.

When the PSA Client call is completed in TF-M, the return result, such as PSA_SUCCESS or a handle, shall be returned from SPE to NSPE via mailbox.

Mailbox synchronization between NSPE and SPE

Synchronization and protection between NSPE and SPE accesses to shared mailbox objects and variables shall be implemented.

When a core accesses shared mailbox objects or variables, proper mechanisms must protect concurrent operations from the other core.

Support of multiple ongoing NS PSA client calls (informative)

If the support of multiple ongoing NS PSA client calls in TF-M is required in dual-core systems, an optional queue can be maintained in TF-M core to store multiple mailbox objects received from NSPE. To identify NS PSA client calls, additional fields can be added in TF-M SPM objects to store the NS PSA Client request identification.

Note that when just a single outstanding PSA client call is allowed, multiple NSPE OS threads can run concurrently and call PSA client functions. The first PSA client call will be processed first, and any other OS threads will be blocked from submitting PSA client calls until the first is completed.

PSA client call handling flow in TF-M

This section provides more details about the flow of PSA client call handing in TF-M.

The sequence of handling PSA Client call request in TF-M is listed as below

1. Platform specific Inter-Processor Communication interrupt handler is triggered after the mailbox event is asserted by NSPE. The interrupt handler shall call spm_handle_interrupt()
2. SPM will send a MAILBOX_INTERRUPT_SIGNAL to ns_agent_mailbox partition
3. ns_agent_mailbox partition deals with the mailbox message(s) which contain(s) the PSA client call information and parameters. Then the PSA client call request is dispatched to dedicated PSA client call handler in TF-M SPM.
4. After the PSA client call is completed, the return value is transmitted to NSPE via mailbox.

Several key modules in the whole process are covered in detail in following sections.

- *Inter-Processor Communication interrupt handler* discusses the Inter-Processor Communication interrupt handler
- *TF-M Remote Procedure Call (RPC) layer* introduces TF-M Remote Procedure Call layer to support dual-core communication.
- *ns_agent_mailbox partition* describes the mailbox agent partition.
- *Return value replying routine in TF-M* proposes the routine to send the return value to NSPE.

Inter-Processor Communication interrupt handler

Platform specific driver shall implement the Inter-Processor Communication interrupt handler to deal with the Inter-Processor Communication interrupt asserted by NSPE. The platform specific interrupt handler shall complete the interrupt operations, such as interrupt EOI or acknowledge.

The interrupt handler shall call spm_handle_interrupt() to notify SPM of the interrupt.

The platform's tfm_peripherals_def.h file shall define a macro MAILBOX_IRQ that identifies the interrupt being used. The platform must also provide a function mailbox_irq_init() that initialises the interrupt as described in[2].

Platform specific driver shall put Inter-Processor Communication interrupt into a proper exception priority, according to system and application requirements. The proper priority setting must guarantee that

- TF-M can respond to a PSA client call request in time according to system and application requirements.
- Other exceptions, which are more latency sensitive or require higher priorities, are not blocked by Inter-Processor Communication interrupt ISR.

The exception priority setting is IMPLEMENTATION DEFINED.

TF-M Remote Procedure Call (RPC) layer

[2] *Secure Interrupt Integration Guide*

This design brings a concept of Remote Procedure Call layer into TF-M.

The RPC layer sits between TF-M SPM and mailbox implementation. The purpose of RPC layer is to decouple mailbox implementation and TF-M SPM and enhance the generality of entire dual-core communication.

The RPC layer provides a set of APIs to TF-M SPM to handle and reply PSA client call from NSPE in dual-core scenario. Please refer to *TF-M RPC definitions to TF-M SPM* for API details. It hides the details of specific mailbox implementation from TF-M SPM. It avoids modifying TF-M SPM to fit mailbox development and changes. It can keep a unified PSA client call process in TF-M SPM in both single Armv8-M scenario and dual core scenario.

The RPC layer defines a set callback functions for mailbox implementation to hook its specific mailbox operations. When TF-M SPM invokes RPC APIs to deal with NSPE PSA client call, RPC layer eventually calls the callbacks to execute mailbox operations. RPC layer also defines a set of PSA client call handler APIs for mailbox implementation. RPC specific client call handlers parse the PSA client call parameters and invoke common TF-M PSA client call handlers. Please refer to *TF-M RPC definitions for mailbox* for the details.

ns_agent_mailbox partition

A partition will be dedicated to interacting with the NSPE through the mailbox. This partition will call tfm_hal_boot_ns_cpu() and tfm_hal_wait_for_ns_cpu_ready() to ensure that the non-secure core is running. It will then initialise the SPE mailbox and enable the IPC interrupt. Once these tasks are complete, it will enter an infinite loop waiting for a MAILBOX_INTERRUPT_SIGNAL signal indicating that a mailbox message has arrived.

Mailbox handling will be done in the context of the ns_agent_mailbox partition, which will make any necessary calls to other partitions on behalf of the non-secure code.

ns_agent_mailbox shall call RPC API tfm_rpc_client_call_handler() to check and handle PSA client call request from NSPE. tfm_rpc_client_call_handler() invokes request handling callback function to eventually execute specific mailbox message handling operations. The mailbox APIs are defined in mailbox design document[Page 1342, 1].

The handling process in mailbox operation consists of the following steps.

1. SPE mailbox fetches the PSA client call parameters from NSPE mailbox. Proper protection and synchronization must be implemented in mailbox to guarantee that the operations are not interfered with by NSPE mailbox operations or Inter-Processor Communication interrupt handler. If a queue is maintained inside TF-M core, SPE mailbox can fetch multiple PSA client calls together into the queue, to save the time of synchronization between two cores.
2. SPE mailbox parses the PSA client call parameters copied from NSPE, including the PSA client call type.
3. The PSA client call request is dispatched to the dedicated TF-M RPC PSA client call handler. The PSA client call request is processed in the corresponding handler.

- For psa_framework_version() and psa_version(), the PSA client call can be completed in the handlers tfm_rpc_psa_framework_version() and tfm_rpc_psa_version() respectively.
- For psa_connect(), psa_call() and psa_close(), the handlers tfm_rpc_psa_connect(), tfm_rpc_psa_call() and tfm_rpc_psa_close() create the PSA message and trigger target Secure partition respectively. The target Secure partition will be woken up to handle the PSA message.

The dual-core scenario and single Armv8-M scenario in TF-M IPC implementation will share the same PSA client call routines inside TF-M SPM. The current handler definitions can be adjusted to be more generic for dual-core scenario and single Armv8-M implementation. Please refer to *Summary of changes to TF-M core/SPM* for details.

If there are multiple NSPE PSA client call requests pending, SPE mailbox can process mailbox messages one by one.

Return value replying routine in TF-M

Diverse PSA client calls can be implemented with different return value replying routines.

- *Replying routine for psa_framework_version() and psa_version()* describes the routine for psa_framework_version() and psa_version().
- *Replying routine for psa_connect(), psa_call() and psa_close()* describes the routine for psa_connect(), psa_call() and psa_close().

Replying routine for psa_framework_version() and psa_version()

For psa_framework_version() and psa_version(), the return value can be directly returned from the dedicated TF-M RPC PSA client call handlers. Therefore, the return value can be directly replied in mailbox handling process.

A compile flag may be defined to enable replying routine via mailbox in dual-core scenario during building.

Replying routine for psa_connect(), psa_call() and psa_close()

For psa_connect(), psa_call() and psa_close(), the PSA client call is completed in the target Secure Partition. The target Secure Partition calls psa_reply() to reply the return value to TF-M SPM. In the SVC handler of psa_reply() in TF-M SPM, TF-M SPM should call TF-M RPC API tfm_rpc_client_call_reply() to return the value to NSPE via mailbox. tfm_rpc_client_call_reply() invokes reply callbacks to execute specific mailbox reply operations. The mailbox reply functions must not trigger a context switch inside SVC handler.

If an error occurs in the handlers, the TF-M RPC handlers, tfm_rpc_psa_call(), tfm_rpc_psa_connect() and tfm_rpc_psa_close(), may terminate and return the error, without triggering the target Secure Partition. The mailbox implementation shall return the error code to NSPE.

Summary of changes to TF-M core/SPM

This section discusses the general changes related to NSPE and SPE communication to current TF-M core/SPM implementations.

The detailed mailbox implementations are not covered in this section. Please refer to mailbox dedicated document[Page 1342, 1]. The platform specific implementations are also not covered in this section, including the Inter-Processor Communication interrupt or its interrupt handler.

Common PSA client call handlers

Common PSA client call handlers shall be extracted from current PSA client call handlers implementation in TF-M. Common PSA client call handlers are shared by both TF-M RPC layer in dual-core scenario and SVCall handlers in single Armv8-M scenario.

TF-M RPC layer

This section describes the TF-M RPC data types and APIs.

- *TF-M RPC definitions to TF-M SPM* lists the data types and APIs to be invoked by TF-M SPM.
- *TF-M RPC definitions for mailbox* lists the data types and APIs to be referred by mailbox implementation

TF-M RPC definitions to TF-M SPM

TFM_RPC_SUCCESS

TFM_RPC_SUCCESS is a general return value to indicate that the RPC operation succeeds.

```
#define TFM_RPC_SUCCESS             (0)
```

TFM_RPC_INVAL_PARAM

TFM_RPC_INVAL_PARAM is a return value to indicate that the input parameters are invalid.

```
#define TFM_RPC_INVAL_PARAM         (INT32_MIN + 1)
```

TFM_RPC_CONFLICT_CALLBACK

Currently one and only one mailbox implementation is supported in dual core communication. This flag indicates that callback functions from one mailbox implementation are already registered and no more implementations are accepted.

```
#define TFM_RPC_CONFLICT_CALLBACK   (INT32_MIN + 2)
```

tfm_rpc_client_call_handler()

TF-M PendSV handler calls this function to handle NSPE PSA client call request.

```
void tfm_rpc_client_call_handler(void);
```

Usage

tfm_rpc_client_call_handler() invokes callback function handle_req() to execute specific mailbox handling. Please note that tfm_rpc_client_call_handler() doesn't return the status of underlying mailbox handling.

tfm_rpc_client_call_reply()

TF-M psa_reply() handler calls this function to reply PSA client call return result to NSPE.

```
void tfm_rpc_client_call_reply(const void *owner, int32_t ret);
```

Parameters

owner	A handle to identify the owner of the PSA client call return value.
ret	PSA client call return result value.

Usage

tfm_rpc_client_call_reply() invokes callback function reply() to execute specific mailbox reply. Please note that tfm_rpc_client_call_reply() doesn't return the status of underlying mailbox reply process.

TF-M RPC definitions for mailbox

Mailbox operations callbacks

This structures contains the callback functions for specific mailbox operations.

```
struct tfm_rpc_ops_t {
    void (*handle_req)(void);
    void (*reply)(const void *owner, int32_t ret);
};
```

tfm_rpc_register_ops()

This function registers underlying mailbox operations into TF-M RPC callbacks.

```
int32_t tfm_rpc_register_ops(const struct tfm_rpc_ops_t *ops_ptr);
```

Parameters

ops_ptr	Pointer to the specific operation structure.

Return

TFM_RPC_SUCCESS	Operations are successfully registered.
Other error code	Fail to register operations.

Usage

Mailbox shall register TF-M RPC callbacks during mailbox initialization, before enabling secure services for NSPE.

Currently one and only one underlying mailbox communication implementation is allowed in runtime.

tfm_rpc_unregister_ops()

This function unregisters underlying mailbox operations from TF-M RPC callbacks.

```
void tfm_rpc_unregister_ops(void);
```

Usage

Currently one and only one underlying mailbox communication implementation is allowed in runtime.

tfm_rpc_psa_framework_version()

TF-M RPC handler for psa_framework_version().

```
uint32_t tfm_rpc_psa_framework_version(void);
```

Return

version	The version of the PSA Framework implementation that is providing the runtime services.

Usage

tfm_rpc_psa_framework_version() invokes common psa_framework_version() handler in TF-M.

tfm_rpc_psa_version()

TF-M RPC handler for psa_version().

```
uint32_t tfm_rpc_psa_version(uint32_t sid);
```

Parameters

sid	RoT Service identity.

Return

PSA_VERSION_NO	The RoT Service is not implemented, or the caller is not permitted to access the service.
> 0	The minor version of the implemented RoT Service.

Usage

tfm_rpc_psa_version() invokes common psa_version() handler in TF-M. The parameters in params shall be prepared before calling tfm_rpc_psa_version().

tfm_rpc_psa_connect()

TF-M RPC handler for psa_connect().

```
psa_status_t tfm_rpc_psa_connect(uint32_t sid,
                                 uint32_t version,
                                 int32_t ns_client_id,
                                 const void *client_data);
```

Parameters

sid	RoT Service identity.
version	The version of the RoT Service.
ns_client_id	Agent representing NS client's identifier.
client_data	Client data, treated as opaque by SPM.

Return

PSA_SUCCESS	Success.
PSA_CONNECTION_	The SPM cannot make the connection at the moment.
Does not return	The RoT Service ID and version are not supported, or the caller is not permitted to access the service.

Usage

tfm_rpc_psa_connect() invokes common psa_connect() handler in TF-M.

tfm_rpc_psa_call()

TF-M RPC handler for psa_call().

```
psa_status_t tfm_rpc_psa_call(psa_handle_t handle, uint32_t control,
                              const struct client_params_t *params,
                              const void *client_data_stateless);
```

Parameters

handle	Handle to the service being accessed.
control	A composited uint32_t value for controlling purpose, containing call types, numbers of in/out vectors and attributes of vectors.
params	Combines the psa_invec and psa_outvec params for the psa_call() to be made, as well as NS agent's client identifier, which is ignored for connection-based services.
client_data_	Client data, treated as opaque by SPM.

Return

PSA_SUCCESS	Success.
Does not return	The call is invalid, or invalid parameters.

Usage

tfm_rpc_psa_call() invokes common psa_call() handler in TF-M. The value of control and parameters in params shall be prepared before calling tfm_rpc_psa_call().

tfm_rpc_psa_close()

TF-M RPC psa_close() handler

```
psa_status_t tfm_rpc_psa_close(psa_handle_t handle);
```

Parameters

handle	A handle to an established connection.

Return

PSA_SUCCESS	Success.
PROGRAMMER ERROR	The call is invalid, or invalid parameters.

Usage

tfm_rpc_psa_close() invokes common psa_close() handler in TF-M.

Other modifications

The following mandatory changes are also required.

- One or more compile flag(s) shall be defined to select corresponding execution routines in dual-core scenario or single Armv8-M scenario during building.

Reference

Copyright (c) 2019-2024 Arm Limited. All Rights Reserved.

Copyright (c) 2020-2023 Cypress Semiconductor Corporation (an Infineon company) or an affiliate of Cypress Semiconductor Corporation. All rights reserved.

Mailbox Design in TF-M on Dual-core System

Author
David Hu

Organization
Arm Limited

Contact
david.hu@arm.com

Introduction

This document proposes a generic design of the mailbox communication for Trusted Firmware-M (TF-M) on a dual-core system. The mailbox functionalities transfer PSA Client requests and results between Non-secure Processing Environment (NSPE) and Secure Processing Environment (SPE).

The dual-core system should satisfy the following requirements

- NSPE and SPE are properly isolated and protected following PSA specifications.
- An Arm M-profile core locates in SPE and acts as the Secure core.
- TF-M runs on the Secure core with platform specific drivers support.
- Inter-Processor Communication hardware module in system for communication between Secure core and Non-secure core. Mailbox communication calls the Inter-Processor Communication to transfer notifications.
- Non-secure memory shared by NSPE and SPE.

Scope

This design document focuses on the mailbox functionalities in NSPE and SPE on a dual-core system. The mailbox functionalities include the initialization of mailbox in NSPE and SPE, and the transfer of PSA Client requests and replies between NSPE and SPE.

Data types and mailbox APIs are defined in this document.

Some details of interactions between mailbox and other modules are specified in other documents. Communication prototype design[1] defines a general communication prototype between NSPE and SPE on a dual-core system. It describes how TF-M interacts with mailbox functionalities and the general requirements on mailbox. Dual-core booting sequence[2] describes a synchronization step of mailbox between NSPE and SPE during system booting.

Organization of the document

- *Mailbox architecture* provides an overview on the mailbox architecture.
- *Mailbox communication for PSA Client calls* discusses about the mailbox communication for PSA Client calls.
- *Mailbox initialization* introduces the initialization of mailbox.
- *Mailbox APIs and data structures* lists mailbox data types and APIs.

Mailbox architecture

The mailbox consists of two parts sitting in NSPE and SPE respectively. NSPE mailbox provides mailbox functionalities in NSPE and SPE mailbox provides mailbox functionalities in TF-M in SPE.

PSA Client APIs called in NSPE are implemented by NSPE mailbox functions on dual-core systems, to send PSA Client request and receive the result. The implementation can vary in diverse NSPE OSs or use cases.

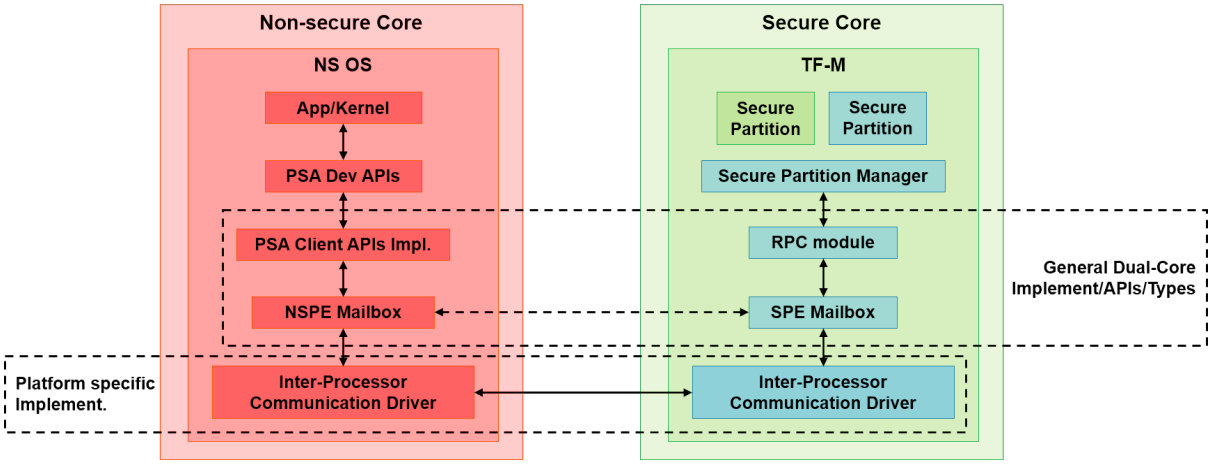
TF-M provides a reference implementation of NSPE mailbox. The NSPE mailbox delivers the PSA Client requests to SPE mailbox. After the PSA Client result is replied from SPE, NSPE mailbox fetches the result and returns it to PSA Client APIs.

NSPE mailbox objects are managed by NSPE mailbox in non-secure memory to hold PSA Client call parameters, return result and other mailbox data. NSPE mailbox relies on platform specific Inter-Process Communication to process notifications between NSPE and SPE.

The SPE mailbox in TF-M receives PSA Client requests from NSPE mailbox. It parses the requests and invokes TF-M Remote Procedure Call (RPC) layer. RPC layer delivers the requests to TF-M core/Secure Partition Manager (SPM). After the PSA Client call is completed, TF-M core/SPM invokes RPC layer to return results to NSPE, via SPE mailbox. SPE mailbox objects are managed by SPE mailbox in secure memory. SPE mailbox relies on platform specific Inter-Process Communication to process notifications between SPE and NSPE.

[1] *Communication prototype between NSPE and SPE in Dual-core systems*
[2] *Booting a Dual-core system*

The architecture is showed in following figure.



Mailbox communication for PSA Client calls

This section describes the transfer of PSA Client request and reply between NSPE and SPE via mailbox.

Mailbox objects

This section lists the mailbox objects required in NSPE and SPE.

NSPE mailbox objects are managed by NSPE mailbox in non-secure memory. But NSPE mailbox objects can be accessed by both NSPE mailbox and SPE mailbox.

SPE mailbox objects are managed by SPE mailbox in secure memory. SPE mailbox objects should be protected from NSPE accesses by system specific isolation.

NSPE Mailbox queue

NSPE mailbox maintains a mailbox queue in non-secure memory. Please refer to the structure definition in *NSPE mailbox queue structure*.

NSPE mailbox queue contains one or more slots. The number of slots should be aligned with that in SPE mailbox queue.

Each slot in NSPE mailbox queue consists of a pair of a mailbox message structure and a mailbox reply structure. Each slot might contain additional fields, such as identification of non-secure task which initiates the PSA Client request. Each slot serves a PSA Client call from non-secure task.

The parameters of PSA Client request are hosted in the mailbox message inside the slot. *Mailbox messages* describes the details of mailbox message.

The mailbox reply structure is used to receive the PSA Client result from SPE. *Mailbox replies* describes the details of mailbox reply.

Mailbox messages

A mailbox message contains the parameters of a PSA Client request from a non-secure task. Please refer to the structure definition in *Mailbox message structure*.

Inside PSA Client API implementation, NSPE mailbox selects an empty mailbox queue slot for the PSA Client request. The parameters of that PSA Client request are organized into the mailbox message belonging to the selected slot. SPE mailbox will parse those parameters from the mailbox message.

More fields can be defined in mailbox message to transfer additional information from NSPE to SPE for processing in TF-M.

Mailbox replies

A mailbox reply structure in non-secure memory receives the PSA Client result replied from SPE mailbox. Please refer to the structure definition in *Mailbox reply structure*.

SPE Mailbox queue

SPE mailbox maintains a mailbox queue to store SPE mailbox objects. Please refer to the structure definition in *SPE mailbox queue structure*.

SPE mailbox queue contains one or more slots. The number of slots should be aligned with that in NSPE mailbox queue. After SPE is notified that a PSA Client request is pending, SPE mailbox can assign an empty slot, copy the corresponding PSA Client call parameters from non-secure memory to that slot and parse the parameters.

Each slot in SPE mailbox queue can contain the following fields

- An optional field to hold mailbox message content copied from non-secure memory.
- Index of NSPE mailbox queue slot containing the mailbox message.
- A handle to the mailbox message. Optional. Identify the owner slot of PSA Client result when multiple mailbox messages are under processing.

More fields can be defined in the slot structure to support mailbox processing in SPE.

Overall workflow

The overall workflow of transferring PSA Client requests and results between NSPE and SPE via mailbox is shown below.

1. Non-secure task initiates a service request by calling PSA Developer APIs, which eventually invoke PSA Client APIs. PSA Client APIs call NSPE mailbox functions to transmit PSA Client call to SPE.
2. NSPE mailbox assigns an empty slot from NSPE mailbox queue for that PSA client call.
3. NSPE mailbox prepares the parameters of PSA Client call in the dedicated mailbox message inside the assigned slot.
4. After the mailbox message is ready, NSPE mailbox invokes platform specific Inter-Processor Communication driver to notify SPE. The notification mechanism of Inter-Processor Communication is platform specific.
5. After the notification is completed, non-secure task waits for the reply from SPE.
6. Platform specific Inter-Processor Communication interrupt for mailbox is asserted in SPE. The interrupt handler activates SPE mailbox to process the request(s).
7. During mailbox processing in TF-M, the handling routine can include the following steps:
    1. SPE mailbox checks and validates NSPE mailbox queue status.
    2. SPE mailbox fetches PSA Client call parameters from NSPE mailbox queue.
    3. SPE mailbox parses the parameters.
    4. SPE mailbox invokes the TF-M RPC APIs to deliver the PSA Client request to TF-M SPM.
    5. The PSA Client call is handled in TF-M SPM and target Secure Partition if necessary.

    If multiple ongoing mailbox messages are pending in the SPE, SPE mailbox can process mailbox messages one by one.

8. After the PSA Client call is completed, TF-M RPC layer notifies SPE mailbox to reply PSA Client result to NSPE.
9. SPE mailbox writes the PSA Client result to the dedicated mailbox reply structure in non-secure memory. The related SPE mailbox objects should be invalidated or cleaned.
10. SPE mailbox notifies NSPE by invoking Inter-Processor Communication driver to send a notification to NSPE. The notification mechanism of Inter-Processor Communication is platform specific.
11. NSPE mailbox is activated to handle the PSA Client result in the mailbox reply structure. Related mailbox objects should be invalidated or cleaned by NSPE mailbox after the return results is extracted out.
12. NSPE mailbox returns the result to PSA Client API implementation. The result is eventually returned to the non-secure task.

The following sections discuss more details of key steps in above sequence.

Mailbox notifications between NSPE and SPE

As shown in *Overall workflow*, NSPE mailbox asserts mailbox notification to trigger SPE to handle PSA Client request. SPE mailbox asserts mailbox notification to notify NSPE that PSA Client result is written. The notification implementation is based on platform specific Inter-Processor Communication.

It is recommended to assign one independent set of Inter-Processor Communication channel to each notification routine respectively, to implement a *full-duplex* notification mechanism between NSPE and SPE. If both notification routines share the same Inter-Processor Communication channel, proper synchronization should be implemented to prevent conflicts between two notification routines.

In SPE, the Inter-Processor Communication interrupt handler should deal with the incoming notification from NSPE and activate the subsequent mailbox handling in SPE. Communication prototype design[Page 1351, 1] defines the behavior of Inter-Processor Communication interrupt handler.

NSPE can implement an interrupt handler or a polling of notification status to handle Inter-Processor Communication notification from SPE.

Implement PSA Client API with NSPE Mailbox

PSA Client APIs are implemented with NSPE mailbox API tfm_ns_mailbox_client_call().

The pseudo code below shows a reference implementation of psa_framework_version().

```
uint32_t psa_framework_version(void)
{
    ...
    int32_t ret;

    ret = tfm_ns_mailbox_client_call(...);
    if (ret != MAILBOX_SUCCESS) {
        version = PSA_VERSION_NONE;
    }

    ...
}
```

tfm_ns_mailbox_client_call() implementation can vary according to usage scenario. TF-M reference implementation provides implementations for NS OS and NS bare metal environment respectively. Refer to *TF-M reference implementation of NSPE mailbox* for details.

As PSA Firmware Framework-M (FF-M) requests, a PSA Client API function should be blocked until the result is returned. To comply with FF-M, NSPE mailbox requires proper mechanism(s) to keep current caller waiting for PSA Client result or an empty mailbox queue slot.

Note: tfm_ns_mailbox_client_call() may trap the current exception in sleep and therefore it must not be called in interrupt service routine.

Refer to *Mailbox APIs and data structures* for details of tfm_ns_mailbox_client_call().

TF-M reference implementation of NSPE mailbox

TF-M NS interface provides a reference implementation of NS mailbox.

This reference implementation defines several NS mailbox HAL APIs. Please refer to *NSPE mailbox HAL APIs* for details.

Integration with NSPE

TF-M reference implementation provides several mailbox build flags to control the integration with NS software.

- TFM_MULTI_CORE_NS_OS

  When integrating NS mailbox with NS OS, such as NS RTOS, that flag can be selected to enable NS OS support in NS mailbox, such as thread management to fulfill thread wait and wake-up. Please refer to *NSPE mailbox RTOS abstraction APIs* for NS OS support details.

  With NS OS support, multiple outstanding PSA Client calls can be supported in NS mailbox when number of mailbox queue slots configured in NUM_MAILBOX_QUEUE_SLOT is greater than 1.

  If TFM_MULTI_CORE_NS_OS is enabled, when a NS client starts a PSA Client call:

  - tfm_ns_mailbox_client_call() selects an empty NSPE mailbox queue slot to organize received PSA client call parameters into a mailbox message.
  - Then it sends those parameters to SPE mailbox and waits for results from SPE. During waiting for the result, the NS client thread may be switched out by NS OS scheduler.
  - When the result arrives, the NS client thread will be woken up inside NS mailbox interrupt handler.
  - The result is then written back to NS client finally.

  When that flag is disabled, NS mailbox runs as being integrated with NS bare metal environment. NS mailbox simply loops mailbox message status while waiting for results.

- TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD

  When TFM_MULTI_CORE_NS_OS is enabled, this flag can be selected to enable another NS mailbox thread model which relies on a NS mailbox dedicated thread.

  – It requires NS OS to create a dedicated thread to perform NS mailbox functionalities. This dedicated thread invokes tfm_ns_mailbox_thread_runner() to handle PSA Client calls. tfm_ns_mailbox_thread_runner() constructs mailbox messages and sends them to SPE mailbox.

  – tfm_ns_mailbox_client_call() sends PSA Client calls to the dedicated mailbox thread. It doesn't directly deal with mailbox messages.

- It also relies on NS OS to provide thread management and inter-thread communication. Please refer to *NSPE mailbox RTOS abstraction APIs* for details.
- It also requires dual-cpu platform to implement NS Inter-Processor Communication interrupts. The interrupt handler invokes tfm_ns_mailbox_wake_reply_owner_isr() to deal with PSA Client call replies and notify the waiting threads.

Multiple outstanding PSA Client call feature

Multiple outstanding PSA Client call feature can enable dual-cpu platform to issue multiple PSA Client calls in NS OS and those PSA Client calls can be served simultaneously.

Without this feature, only a single PSA Client call can be issued and served. A new PSA Client call cannot be started until the previous one is completed.

When multiple outstanding PSA Client call feature is enabled, while a NS application is waiting for its PSA Client result, another NS application can be switched in by NS OS to prepare another PSA Client call or deal with its PSA client result. It can decrease the CPU idle time of waiting for PSA Client call completion.

If multiple NS applications request secure services in NS OS, it is recommended to enable this feature.

To implement this feature in NS OS:

- Platform should set the number of mailbox queue slots in NUM_MAILBOX_QUEUE_SLOT in platform's config.cmake. It will use more data area with multiple mailbox queue slots.

  NSPE and SPE share the same NUM_MAILBOX_QUEUE_SLOT value.

- Enable TFM_MULTI_CORE_NS_OS

  For more details, refer to *TFM_MULTI_CORE_NS_OS*.

  TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD can be enabled to select another NS mailbox working model. See *TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD* for details.

Critical section protection between cores

Proper protection should be implemented to protect the critical accesses to shared mailbox resources. The critical sections can include atomic reading and modifying NSPE mailbox queue status, slot status and other critical operations.

The implementation should protect a critical access to those shared mailbox resource from corruptions caused by accesses from the peer core. SPE mailbox also accesses NSPE mailbox queue. Therefore, it is essential to implement synchronization or protection on NSPE mailbox queue between Secure core and Non-secure core. NSPE mailbox and SPE mailbox define corresponding critical section protection APIs respectively. The implementation of those APIs can be platform specific. Please see more details in *NSPE mailbox APIs* and *SPE mailbox APIs*.

It is recommended to rely on both hardware and software to implement the synchronization and protection.

Protection of local mailbox objects can be implemented as static functions inside NSPE mailbox and SPE mailbox.

Mailbox handling in TF-M

According to communication prototype design[Page 1351, 1], mailbox implementation should invoke tfm_rpc_register_ops() to hook its operations to TF-M RPC module callbacks during initialization. Mailbox message handling should call TF-M RPC PSA Client call handlers to deliver PSA Client request to TF-M SPM.

If multiple outstanding NS PSA Client calls should be supported, TF-M SPM can store the mailbox message handle in a specific field in PSA message structure to identify the mailbox message,

while creating a PSA message. While replying the PSA Client result, TF-M SPM can extract the mailbox message handle from PSA message and pass it back to mailbox reply function. SPE mailbox can identify which mailbox message is completed according to the handle and write the result to corresponding NSPE mailbox queue slot.

Platform specific Inter-Processor Communication interrupt handler in SPE should call spm_handle_interrupt() to notify SPM of the interrupt. SPM will then send the MAILBOX_INTERRUPT_SIGNAL signal to the ns_agent_mailbox partition, which will call tfm_rpc_client_call_handler().

Mailbox initialization

It should be guaranteed that NSPE mailbox should not initiate PSA Client request until SPE mailbox initialization completes. Refer to dual-core booting sequence[Page 1351, 2] for more details on the synchronization between NSPE and SPE during booting.

In current design, the base address of NSPE mailbox queue should be pre-defined and shared between NSPE mailbox and SPE mailbox.

SPE mailbox initialization

The SPE mailbox queue memory should be allocated before calling tfm_mailbox_init(). tfm_mailbox_init() initializes the memory and variables. tfm_mailbox_init() calls tfm_mailbox_hal_init() to perform platform specific initialization. The base address of NSPE mailbox queue can be received via tfm_mailbox_hal_init().

SPE mailbox dedicated Inter-Processor Communication initialization can also be enabled during SPE mailbox initialization.

After SPE mailbox initialization completes, SPE notifies NSPE that SPE mailbox functionalities are ready.

NSPE mailbox initialization

The NSPE mailbox queue memory should be allocated before calling tfm_ns_mailbox_init(). tfm_ns_mailbox_init() initializes the memory and variables. tfm_ns_mailbox_init() calls tfm_ns_mailbox_hal_init() to perform platform specific initialization. The base address of NSPE mailbox queue can be passed to SPE mailbox via tfm_ns_mailbox_hal_init().

NSPE mailbox dedicated Inter-Processor Communication initialization can also be enabled during NSPE mailbox initialization.

Mailbox APIs and data structures

Data types

Constants

MAILBOX_SUCCESS

MAILBOX_SUCCESS is a generic return value to indicate success of mailbox operation.

```
#define MAILBOX_SUCCESS         (0)
```

MAILBOX_QUEUE_FULL

MAILBOX_QUEUE_FULL is a return value from mailbox function if mailbox queue is full.

```
#define MAILBOX_QUEUE_FULL      (INT32_MIN + 1)
```

MAILBOX_INVAL_PARAMS

MAILBOX_INVAL_PARAMS is a return value from mailbox function if any parameter is invalid.

```
#define MAILBOX_INVAL_PARAMS    (INT32_MIN + 2)
```

MAILBOX_NO_PERMS

MAILBOX_NO_PERMS is a return value from mailbox function if the caller doesn't own a proper permission to execute the operation.

```
#define MAILBOX_NO_PERMS        (INT32_MIN + 3)
```

MAILBOX_NO_PEND_EVENT

MAILBOX_NO_PEND_EVENT is a return value from mailbox function if the expected event doesn't occur yet.

```
#define MAILBOX_NO_PEND_EVENT   (INT32_MIN + 4)
```

MAILBOX_CHAN_BUSY

MAILBOX_CHAN_BUSY is a return value from mailbox function if the underlying Inter-Processor Communication resource is busy.

```
#define MAILBOX_CHAN_BUSY       (INT32_MIN + 5)
```

MAILBOX_CALLBACK_REG_ERROR

MAILBOX_CALLBACK_REG_ERROR is a return value from mailbox function if the registration of mailbox callback functions failed.

```
#define MAILBOX_CALLBACK_REG_ERROR      (INT32_MIN + 6)
```

MAILBOX_INIT_ERROR

MAILBOX_INIT_ERROR is a return value from mailbox function if the mailbox initialization failed.

```
#define MAILBOX_INIT_ERROR      (INT32_MIN + 7)
```

MAILBOX_GENERIC_ERROR

MAILBOX_GENERIC_ERROR indicates mailbox generic errors which cannot be indicated by the codes above.

```
#define MAILBOX_GENERIC_ERROR    (INT32_MIN + 8)
```

PSA Client API types

The following constants define the PSA Client API type values shared between NSPE and SPE

```
#define MAILBOX_PSA_FRAMEWORK_VERSION       (0x1)
#define MAILBOX_PSA_VERSION                 (0x2)
#define MAILBOX_PSA_CONNECT                 (0x3)
#define MAILBOX_PSA_CALL                    (0x4)
#define MAILBOX_PSA_CLOSE                   (0x5)
```

Mailbox message structure

psa_client_params_t lists the parameters passed from NSPE to SPE required by a PSA Client call.

```
struct psa_client_params_t {
    union {
        struct {
            uint32_t        sid;
        } psa_version_params;

        struct {
            uint32_t        sid;
            uint32_t        minor_version;
        } psa_connect_params;

        struct {
            psa_handle_t    handle;
            int32_t         type;
            const psa_invec *in_vec;
            size_t          in_len;
            psa_outvec      *out_vec;
            size_t          out_len;
        } psa_call_params;

        struct {
            psa_handle_t    handle;
        } psa_close_params;
    };
};
```

The following structure describe a mailbox message and its members.

- call_type indicates the PSA Client API type.
- params stores the PSA Client call parameters.
- client_id records the client ID of the non-secure client. Optional. It is used to identify the non-secure tasks in TF-M when NSPE OS enforces non-secure task isolation.

```
struct mailbox_msg_t {
    uint32_t                    call_type;
    struct psa_client_params_t  params;

    int32_t                     client_id;
};
```

Mailbox reply structure

This structure describes a mailbox reply structure, which is managed by NSPE mailbox in non-secure memory.

```
struct mailbox_reply_t {
    int32_t    return_val;
    const void *owner;
    int32_t    *reply;
    uint8_t    *woken_flag;
};
```

Mailbox queue status bitmask

mailbox_queue_status_t defines a bitmask to indicate a status of slots in mailbox queues.

```
typedef uint32_t    mailbox_queue_status_t;
```

NSPE mailbox queue structure

ns_mailbox_slot_t defines a non-secure mailbox queue slot.

```
/* A single slot structure in NSPE mailbox queue */
struct ns_mailbox_slot_t {
    struct mailbox_msg_t    msg;
    struct mailbox_reply_t reply;
};
```

ns_mailbox_queue_t describes the NSPE mailbox queue and its members in non-secure memory.

- empty_slots is the bitmask of empty slots.
- pend_slots is the bitmask of slots whose PSA Client call is not replied yet.
- replied_slots is the bitmask of slots whose PSA Client result is returned but not extracted yet.
- queue is the NSPE mailbox queue of slots.
- is_full indicates whether NS mailbox queue is full.

```
struct ns_mailbox_queue_t {
    mailbox_queue_status_t    empty_slots;
    mailbox_queue_status_t    pend_slots;
    mailbox_queue_status_t    replied_slots;

    struct ns_mailbox_slot_t queue[NUM_MAILBOX_QUEUE_SLOT];

    bool                      is_full;
};
```

SPE mailbox queue structure

secure_mailbox_slot_t defines a single slot structure in SPE mailbox queue.

- ns_slot_idx records the index of NSPE mailbox slot containing the mailbox message under processing. SPE mailbox determines the reply structure address according to this index.
- msg_handle contains the handle to the mailbox message under processing. The handle can be delivered to TF-M SPM while creating PSA message to identify the mailbox message.

```
/* A handle to a mailbox message in use */
typedef int32_t     mailbox_msg_handle_t;

struct secure_mailbox_slot_t {
    uint8_t              ns_slot_idx;
    mailbox_msg_handle_t msg_handle;
};
```

secure_mailbox_queue_t describes the SPE mailbox queue in secure memory.

- empty_slots is the bitmask of empty slots.
- queue is the SPE mailbox queue of slots.
- ns_queue stores the address of NSPE mailbox queue structure.

```
struct secure_mailbox_queue_t {
    mailbox_queue_status_t       empty_slots;

    struct secure_mailbox_slot_t queue[NUM_MAILBOX_QUEUE_SLOT];
```

(continues on next page)

(continued from previous page)

```
    /* Base address of NSPE mailbox queue in non-secure memory */
    struct ns_mailbox_queue_t     *ns_queue;
};
```

NSPE mailbox APIs

NSPE mailbox interface APIs

APIs defined in this section are called by NS software and PSA Client APIs implementations.

tfm_ns_mailbox_init()

This function initializes NSPE mailbox.

```
int32_t tfm_ns_mailbox_init(struct ns_mailbox_queue_t *queue);
```

Parameters

queue	The base address of NSPE mailbox queue.

Return

MAILBOX_SUCCESS	Initialization succeeds.
Other return codes	Initialization fails with an error code.

Usage

tfm_ns_mailbox_init() invokes tfm_ns_mailbox_hal_init() to complete platform specific mailbox and Inter-Processor Communication initialization. The non-secure memory area for NSPE mailbox queue structure should be statically or dynamically pre-allocated before calling tfm_ns_mailbox_init().

tfm_ns_mailbox_client_call()

This function sends the PSA Client request to SPE, waits and fetches PSA Client result.

```
int32_t tfm_ns_mailbox_client_call(uint32_t call_type,
                                   const struct psa_client_params_t *params,
                                   int32_t client_id,
                                   int32_t *reply);
```

Parameters

call_type	Type of PSA Client call
params	Address of PSA Client call parameters structure.
client_id	ID of non-secure task.
reply	The NS client task private buffer written with PSA Client result

Return

MAILBOX_SUCCESS	PSA Client call is completed successfully.
Other return code	Operation failed with an error code.

Usage

If TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD is enabled, tfm_ns_mailbox_client_call() will forward PSA Client calls to the dedicated mailbox thread via NS OS message queue. Otherwise, tfm_ns_mailbox_client_call() directly deals with PSA Client calls and perform NS mailbox functionalities.

tfm_ns_mailbox_thread_runner()

This function handles PSA Client call inside a dedicated NS mailbox thread. It constructs mailbox messages and transmits them to SPE mailbox.

```
void tfm_ns_mailbox_thread_runner(void *args);
```

Parameters

args	The pointer to the structure of PSA Client call parameters.

Usage

tfm_ns_mailbox_thread_runner() should be executed inside the dedicated mailbox thread.

Note: tfm_ns_mailbox_thread_runner() is implemented as an empty function when TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD is disabled.

tfm_ns_mailbox_wake_reply_owner_isr()

This function wakes up the owner task(s) of the returned PSA Client result(s).

```
int32_t tfm_ns_mailbox_wake_reply_owner_isr(void);
```

Return

MAILBOX_SUCCESS	The tasks of replied mailbox messages were found and wake-up signals were sent.
MAILBOX_NO_PEND_EVENT	No replied mailbox message is found.
Other return codes	Operation failed with an error code

Usage

tfm_ns_mailbox_wake_reply_owner_isr() should be called from platform specific Inter-Processor Communication interrupt handler.

Note: tfm_ns_mailbox_wake_reply_owner_isr() is implemented as a dummy function when TFM_MULTI_CORE_NS_OS is disabled.

NSPE mailbox HAL APIs

The HAL APIs defined in this section should be implemented by platform-specific implementation.

This section describes a *reference design* of NSPE mailbox HAL APIs. Developers can define and implement different APIs.

tfm_ns_mailbox_hal_init()

This function executes platform-specific NSPE mailbox initialization.

```
int32_t tfm_ns_mailbox_hal_init(struct ns_mailbox_queue_t *queue);
```

Parameters

queue	The base address of NSPE mailbox queue.

Return

MAILBOX_SUCCESS	Initialization succeeds.
Other return codes	Initialization fails with an error code.

Usage

tfm_ns_mailbox_hal_init() performs platform specific mailbox and Inter-Processor Communication initialization. tfm_ns_mailbox_hal_init() can also share the address of NSPE mailbox queue with SPE mailbox via platform specific implementation.

tfm_ns_mailbox_hal_notify_peer()

This function invokes platform specific Inter-Processor Communication drivers to send notification to SPE.

```
int32_t tfm_ns_mailbox_hal_notify_peer(void);
```

Return

MAILBOX_SUCCESS	The operation completes successfully.
Other return codes	Operation fails with an error code.

Usage

tfm_ns_mailbox_hal_notify_peer() should be implemented by platform specific Inter-Processor Communication drivers.

tfm_ns_mailbox_hal_notify_peer() should not be exported outside NSPE mailbox.

tfm_ns_mailbox_hal_enter_critical()

This function enters the critical section of NSPE mailbox queue access.

```
void tfm_ns_mailbox_hal_enter_critical(void);
```

Usage

NSPE mailbox invokes tfm_ns_mailbox_hal_enter_critical() before entering critical section of NSPE mailbox queue. tfm_ns_mailbox_hal_enter_critical() implementation is platform specific.

tfm_ns_mailbox_hal_enter_critical() should not be called in any interrupt service routine.

tfm_ns_mailbox_hal_exit_critical()

This function exits the critical section of NSPE mailbox queue access.

```
void tfm_ns_mailbox_hal_exit_critical(void);
```

Usage

NSPE mailbox invokes tfm_ns_mailbox_hal_exit_critical() after exiting critical section of NSPE mailbox queue. tfm_ns_mailbox_hal_exit_critical() implementation is platform specific.

tfm_ns_mailbox_hal_exit_critical() should not be called in any interrupt service routine.

tfm_ns_mailbox_hal_enter_critical_isr()

This function enters the critical section of NSPE mailbox queue access in an IRQ handler.

```
void tfm_ns_mailbox_hal_enter_critical(void);
```

Usage

NSPE mailbox invokes tfm_ns_mailbox_hal_enter_critical_isr() before entering critical section of NSPE mailbox queue in an IRQ handler. tfm_ns_mailbox_hal_enter_critical_isr() implementation is platform specific.

tfm_ns_mailbox_hal_exit_critical_isr()

This function exits the critical section of NSPE mailbox queue access in an IRQ handler

```
void tfm_ns_mailbox_hal_exit_critical_isr(void);
```

Usage

NSPE mailbox invokes tfm_ns_mailbox_hal_exit_critical_isr() after exiting critical section of NSPE mailbox queue in an IRQ handler. tfm_ns_mailbox_hal_exit_critical_isr() implementation is platform specific.

NSPE mailbox RTOS abstraction APIs

The APIs defined in this section should be implemented by RTOS-specific implementation when TFM_MULTI_CORE_NS_OS is enabled.

Note: If TFM_MULTI_CORE_NS_OS is set to OFF, the following APIs are defined as dummy functions or empty functions.

---

tfm_ns_mailbox_os_lock_init()

This function initializes the multi-core lock for synchronizing PSA client call(s). The actual implementation depends on the non-secure use scenario.

```
int32_t tfm_ns_mailbox_os_lock_init(void);
```

Return

MAILBOX_SUCCESS	Initialization succeeded.
MAILBOX_GENERIC_ERROR	Initialization failed.

Usage

tfm_ns_mailbox_init() invokes this function to initialize the lock. If TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD is enabled, tfm_ns_mailbox_os_lock_init() is defined as a dummy one.

tfm_ns_mailbox_os_lock_acquire()

This function acquires the multi-core lock for synchronizing PSA client call(s). The actual implementation depends on the non-secure use scenario.

```
int32_t tfm_ns_mailbox_os_lock_acquire(void);
```

Return

MAILBOX_SUCCESS	Succeeded to acquire the lock.
MAILBOX_GENERIC_ERROR	Failed to acquire the lock.

Usage

tfm_ns_mailbox_client_call() invokes this function to acquire the lock when TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD is disabled If TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD is enabled, tfm_ns_mailbox_os_lock_acquire() is defined as a dummy one.

tfm_ns_mailbox_os_lock_release()

This function releases the multi-core lock for synchronizing PSA client call(s). The actual implementation depends on the non-secure use scenario.

```
int32_t tfm_ns_mailbox_os_lock_release(void);
```

Return

MAILBOX_SUCCESS	Succeeded to release the lock.
MAILBOX_GENERIC_ERROR	Failed to release the lock.

Usage

tfm_ns_mailbox_client_call() invokes this function to release the lock when TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD is disabled If TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD is enabled, tfm_ns_mailbox_os_lock_release() is defined as a dummy one.

tfm_ns_mailbox_os_get_task_handle()

This function gets the handle of the current non-secure task executing mailbox functionalities.

```
void *tfm_ns_mailbox_os_get_task_handle(void);
```

Return

Task handle	The non-secure task handle waiting for PSA Client result.

tfm_ns_mailbox_os_wait_reply()

This function performs use scenario and NS OS specific waiting mechanism to wait for the reply of the specified mailbox message to be returned from SPE.

```
void tfm_ns_mailbox_os_wait_reply(void);
```

Usage

The PSA Client API implementations call tfm_ns_mailbox_os_wait_reply() to fall into sleep to wait for PSA Client result.

tfm_ns_mailbox_os_wake_task_isr()

This function wakes up the dedicated task which is waiting for PSA Client result, via RTOS-specific wake-up mechanism.

```
void tfm_ns_mailbox_hal_wait_reply(const void *task_handle);
```

Parameters

task_handle	The handle to the task to be woken up.

tfm_ns_mailbox_os_mq_create()

This function creates and initializes a NS OS message queue.

```
void *tfm_ns_mailbox_os_mq_create(ize_t msg_size, uint8_t msg_count);
```

Parameters

msg_size	The maximum message size in bytes.
msg_count	The maximum number of messages in queue.

Return

message queue handle	The handle of the message queue created, or NULL in case of error.

Usage

If TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD is disabled, tfm_ns_mailbox_os_mq_create() is defined as a dummy one.

tfm_ns_mailbox_os_mq_send()

This function sends PSA Client call request via NS OS message queue.

```
int32_t tfm_ns_mailbox_os_mq_send(void *mq_handle,
                                  const void *msg_ptr);
```

Parameters

mq_handle	The handle of message queue.
msg_ptr	The pointer to the message to be sent.

Return

MAILBOX_SUCCESS	The message is successfully sent.
Other return code	Operation fails with an error code.

Usage

If TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD is disabled, tfm_ns_mailbox_os_mq_send() is defined as a dummy one.

tfm_ns_mailbox_os_mq_receive()

This function receives PSA Client call requests via NS OS message queue.

```
int32_t tfm_ns_mailbox_os_mq_receive(void *mq_handle,
                                     void *msg_ptr);
```

Parameters

mq_handle	The handle of message queue.
msg_ptr	The pointer to buffer for message to be received.

Return

MAILBOX_SUCCESS	A message is successfully received.
Other return code	Operation fails with an error code.

Usage

The buffer size must be large enough to contain the request whose size is set in msg_size `` in ``tfm_ns_mailbox_os_mq_create().

If TFM_MULTI_CORE_NS_OS_MAILBOX_THREAD is disabled, tfm_ns_mailbox_os_mq_receive() is defined as a dummy one.

Note: The function caller should be blocked until a PSA Client call request is received from message queue, unless a fatal error occurs.

SPE mailbox APIs

SPE mailbox interface APIs

The APIs defined in this section are called in TF-M routines and platform specific secure interrupt handler.

tfm_mailbox_handle_msg()

This function completes the handling of mailbox messages from NSPE.

```
int32_t tfm_mailbox_handle_msg(void);
```

Return

MAILBOX_SUCCESS	The operation completes successfully.
Other return codes	Operation fails with an error code.

Usage

tfm_mailbox_handle_msg() is registered to RPC callback function handle_req.

tfm_mailbox_handle_msg() executes the following tasks:

- Check NSPE mailbox queue status.
- Copy mailbox message(s) from NSPE. Optional.
- Checks and validations if necessary
- Parse mailbox message
- Call TF-M RPC APIs to pass PSA Client request to TF-M SPM.

tfm_mailbox_reply_msg()

This function replies the PSA Client result to NSPE.

```
int32_t tfm_mailbox_reply_msg(mailbox_msg_handle_t handle, int32_t reply);
```

Parameters

handle	The handle to mailbox message related to the PSA Client result.
reply	The PSA Client result value to be replied.

Return

MAILBOX_SUCCESS	The operation completes successfully.
Other return codes	Operation fails with an error code.

Usage

tfm_mailbox_reply_msg() is registered to RPC callback reply. It is invoked inside handler of psa_reply() to return the PSA Client result to NSPE.

handle determines which mailbox message in SPE mailbox queue contains the PSA Client call. If handle is set as MAILBOX_MSG_NULL_HANDLE, the return result is replied to the mailbox message in the first SPE mailbox queue slot.

tfm_mailbox_init()

This function initializes SPE mailbox.

```
int32_t tfm_mailbox_init(void);
```

Return

MAILBOX_SUCCESS	Initialization succeeds.
Other return codes	Initialization failed with an error code.

Usage

tfm_mailbox_init() invokes tfm_mailbox_hal_init() to execute platform specific initialization.

SPE mailbox HAL APIs

tfm_mailbox_hal_notify_peer()

This function invokes platform specific Inter-Processor Communication drivers to send notification to NSPE.

```
int32_t tfm_mailbox_hal_notify_peer(void);
```

Return

MAILBOX_SUCCESS	The operation completes successfully.
Other return codes	Operation fails with an error code.

Usage

tfm_mailbox_hal_notify_peer() should be implemented by platform specific Inter-Processor Communication drivers.

tfm_mailbox_hal_notify_peer() should not be exported outside SPE mailbox.

tfm_mailbox_hal_init()

This function is implemented by platform support in TF-M. It completes platform specific mailbox initialization, including receiving the the address of NSPE mailbox queue and Inter-Processor Communication initialization.

```
int32_t tfm_mailbox_hal_init(struct secure_mailbox_queue_t *s_queue);
```

Parameters

s_queue	The base address of SPE mailbox queue.

Return

MAILBOX_SUCCESS	Initialization succeeds.
Other return codes	Initialization failed with an error code.

tfm_mailbox_hal_enter_critical()

This function enters the critical section of NSPE mailbox queue access in SPE.

```
void tfm_mailbox_hal_enter_critical(void);
```

Usage

SPE mailbox invokes tfm_mailbox_hal_enter_critical() before entering critical section of NSPE mailbox queue. tfm_mailbox_hal_enter_critical() implementation is platform specific.

tfm_mailbox_hal_enter_critical() can be called in an interrupt service routine.

tfm_mailbox_hal_exit_critical()

This function exits from the critical section of NSPE mailbox queue access in SPE.

```
void tfm_mailbox_hal_exit_critical(void);
```

Usage

SPE mailbox invokes tfm_mailbox_hal_exit_critical() when exiting from critical section of NSPE mailbox queue. tfm_mailbox_hal_exit_critical() implementation is platform specific.

tfm_mailbox_hal_exit_critical() can be called in an interrupt service routine.

Reference

Copyright (c) 2019-2024 Arm Limited. All Rights Reserved. Copyright (c) 2022 Cypress Semiconductor Corporation. All rights reserved.

Mailbox NS Agent Design Update

Organization
Arm Limited

Contact
tf-m@lists.trustedfirmware.org

Background

The SPE component that maintains the non-secure clients' request is called 'NS Agent' in TF-M. Besides the Trustzone-based isolation mechanism, there is one other isolation mechanism that implements individual PEs in physically isolated cores respectively. NSPE and SPE transfer non-secure client requests via inter-processor communication based on mailboxes. The component that handles inter-processor communication messages is called Mailbox NS Agent.

Note: There may be hardware components and software solutions containing 'mailbox' in their names. The concept mailbox in this document represent the mechanism described above, which is not referring to the external concepts.

When the first version Mailbox NS Agent was introduced, the generic FF-M interrupt handling was not ready. Hence a customized solution Multiple Core is implemented. This customized implementation:

- Perform customized operations on SPM internal data in a deferred interrupt handler.
- Process mailbox operations as special cases in SPM common logic.

These behaviours couple SPM tightly with mailbox logic, which bring issues for maintenance. To address the issue, an updated design shall:

- Make SPM manage other components in a unified way (For example, it is simpler for SPM if all non-SPM components under the IPC model act as processes).
- Can use FF-M compliant interrupt mechanism and APIs.

Following the above guidelines makes the Mailbox NS Agent work like a partition. The agent has an endless loop and waits for signals, calls FF-M API based on the parsing result on the communication messages. But there are still issues after looking closer to the requirements of the agent:

- SPM treats FF-M Client API caller's ID as the client ID. While the mailbox NS agent may represent multiple non-secure clients. Hence it needs to tell SPM which non-secure client it is representing, and the default FF-M Client API does not have such capability.
- FF-M Client API blocks caller before the call is replied; while the mailbox NS Agent needs to respond to the non-secure interrupts in time. Blocking while waiting for a reply may cause the non-secure communication message not to be handled in time.

Extra design items need to be added to address the issues.

Design Update

The below figure shows the overall design to cover various component types. NS Agents are the implementation-defined components that provide FF-M compliant Client API to the non-secure clients. Hence from the view of the non-secure clients, the FF-M client API behaviour follows the FF-M definition. And NS Agent needs customization in SPM since it has extra requirements compared to a generic secure partition.

Note: 3 non-SPM component types here: FF-M-compliant Secure Partition (aka partition), Trustzone-based NS Agent (aka Trustzone NS Agent) and mailbox-based NS Agent (aka Mailbox NS Agent). Trustzone NS Agent is mentioned here for the comparison purpose. The implementation details for this NS agent type is not introduced here.

To make the programming model close to the FF-M compliance, the Mailbox NS Agent is designed as:

- Working like a standard Secure Partition under the IPC model, has one single thread, can call FF-M standard API.

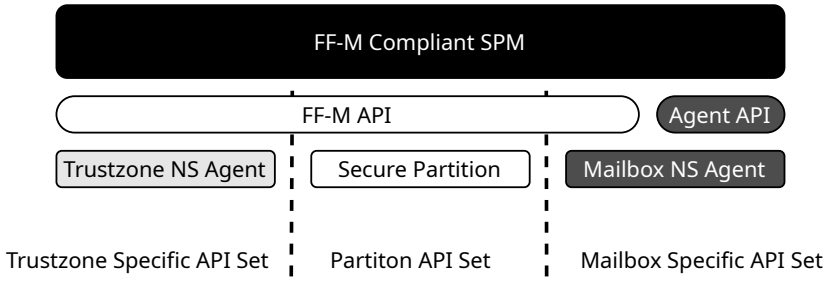


Fig. 4: Component types and the callable API set

- Having a manifest file to describe the attributes and resources and a positive value Partition ID in the manifest.

Services rely on the client_id to apply policy-checking, hence SPM needs to know which client_id the mailbox NS Agent is representing when mailbox API is calling Client API. The standard API treats the caller as the client of the service, which means that a specific API is required to support identifying the represented non-secure client. SPM sets the non-secure client_id into the message right at the moment the message is going to be sent. Before this point, SPM performs the call based on the agent's ID.

This specifc Agent API is non-blocking, unlike the standard FF-M Client APIs. This can improve the communication efficiency between NS clients and mailbox NS agents. With this mechanism, extra signals and APIs for message acknowledges are also required.

Note: A standard Secure Partition gets errors when calling the Agent API.

Updated programming interfaces

These Client APIs are expanded from the standard Client APIs:

- agent_psa_connect() is extended from psa_connect().
- agent_psa_close() is extended from psa_close().
- agent_psa_call() is extended from psa_call().

And to cooperate with the changed behaviour of these APIs, extra defined signals and types are also involved.

Note: Namespace agent is involved for NS Agent callable API; namespace tfm is involved for TF-M specific concepts. Even though agent is TF-M specific at the current stage, it is proposed to be a common concept for general FF-M compliant implementations, hence assigning agent for proposed API and data structures.

Input and output vectors

When non-secure clients call psa_call(), a mailbox message containing psa_call() parameters is delivered into Mailbox NS Agent and the agent needs to extract parameters from the message and then call agent_psa_call(). Revisit the psa_call() prototype to see the parameters:

```
psa_status_t psa_call(psa_handle_t handle,
                      int32_t type,
                      const psa_invec *in_vec,
                      size_t in_len,
```

(continues on next page)

(continued from previous page)

```
                psa_outvec *out_vec,
                size_t out_len);
```

Interface agent_psa_call() has 4 arguments only, to avoid ABI complexity when more than 4 arguments get involved. Then input and output vectors must be squashed into a new type to achieve this squashing. There are several scenarios to be considered:

- In the shared-memory-based mailbox scheme, the non-secure interface layer puts psa_invec and psa_outvec instances and their members in the shared memory. The pointers of these items are delivered to Mailbox NS Agent for the agent's referencing. All vectors are non-secure in this case.
- Still, in the shared-memory-based mailbox scheme, but this time not all parameters are prepared by non-secure clients. Some of the items are allocated by the agent. Secure and non-secure vectors get mixed in this case.
- In the scheme that a memory address can not be delivered (A serial interface e.g.), Mailbox NS Agent allocates psa_invec and psa_outvec in local memory to collect pointers and sizes of received buffers. All the vectors are secure in this case.

Based on these scenarios, and the case that an agent might access depending services with the agent itself's identifier and secure vectors, a straight conclusion is that the memory source information - secure or non-secure - for input and output vectors and their pointing memories need to be indicated by the agent so that SPM can perform a memory check towards given vectors.

Most of the SPE platforms have the capability to identify if a given memory pointer is secure or not, which makes this indication look duplicated. But the indication is necessary for these scenarios:

- The SPE platform identifies the memory source by the address mapped into SPE's address space, but this mapping happens inside the agent instead of SPM. It is the agent who receives the mailbox data and maps it, so it knows which addresses need mapping and the others do not.
- The SPE platform just does not have such capability. The addresses from the mailbox can be treated as non-secure always but there are cases that the agent itself needs to access services with its own memory instead of representing the non-secure clients.

To cover the above mentioned scenarios, guidelines are listed for input and output vector processing:

- The agent needs to tell SPM where vectors and descriptors come from, to assist SPM performs a proper memory checking. The source information is encoded in the parameter control.
- When SPE platforms have the capability to identify the memory sources, platforms can decide whether to skip the indication or not, in the HAL.

A composition type is created for packing the vectors:

```
struct client_params_t {
  int32_t          ns_client_id_stateless;
  const psa_invec *p_invecs;
  psa_outvec       *p_outvecs;
};
```

ns_client_id_stateless indicates the non-secure client id when the client is accessing a stateless service. This member is ignored if the target service is a connection-based one.

Note: The vectors and non-secure client ID are recorded in the internal handle. Hence it is safe to claim client_params_t instance as local variable.

Agent-specific Client API

agent_psa_connect() and agent_psa_close() are the APIs added to support agent forwarding NS requests.

```
psa_handle_t agent_psa_connect(uint32_t sid, uint32_t version,
                               int32_t ns_client_id, const void *client_data);
psa_status_t agent_psa_close(psa_handle_t handle, int32_t ns_client_id);
```

One extra parameter ns_client_id added to tell SPM which NS client the agent is representing when API gets called. It is recorded in the handle association data in SPM and requires to be a negative value; ZERO or positive values are invalid non-secure client IDs, SPM does not use these invalid IDs in the message. Instead, it puts the agent's ID into the messaging in this case. This mechanism can provide chances for the agents calling APIs for their own service accessing and API works asynchronously.

As mentioned, the standard FF-M Client service accessing API are blocked until the IPC message gets replied to. While this API returns immediately without waiting for acknowledgement. Unless an error occurred, these agent-specific API returns PSA_SUCCESS always. The replies for these access requests are always fetched initiative by the agent with a psa_get().

```
psa_status_t agent_psa_call(psa_handle_t handle, uint32_t control,
                            const struct client_param_t    *params,
                            const void    *client_data_stateless);
```

Compared to the standard psa_call(), this API:

- Squashes the psa_invec, psa_outvec, and ns_client_id_stateless into parameter params.
- One extra parameter client_data_stateless for agent_psa_call() stands for the auxiliary data added. This member is ignored for connection-based services because agent_psa_connect() already assigned one in the connected handle.
- Has a composite argument control.

The encoding scheme for control:

Bit(s)	Name	Description
27	NSIV	*1*: Input vectors in non-secure memory. *0*: Secure memory.
24-26	IVNUM	Number of input vectors.
19	NSOV	*1*: Output vectors in non-secure memory. *0*: Secure memory.
16-18	OVNUM	Number of output vectors.
0-15	type	signed 16-bit service *type*.

Note: control is a 32-bit unsigned integer and bits not mentioned in the table are reserved for future usage.

Agent-specific signal

To cooperate with the agent-specific API, one extra acknowledgement signal is defined:

```
#define ASYNC_MSG_REPLY          (0x00000004u)
```

This signal can be sent to agent type component only. An agent can call psa_get() with this signal to get one acknowledged message. This signal is cleared when all queued messages for the agent have been retrieved using psa_get(). SPM assembles the information into agent provided message object. For the stateless handle, the internal handle object is freed after this psa_get() call. The agent can know what kind of message is acknowledged by the type member in the

psa_msg_t, and the client_data passed in is put in member rhandle. If no 'ASYNC_MSG_REPLY' signals pending, calling psa_get() gets panic.

Code Example

```
/*
 * The actual implementation can change this ___customized_t freely, or
 * discard this type and apply some in-house mechanism - the example
 * here is to introduce how an agent works only.
 */
struct ___customized_t {
    int32_t      type;
    int32_t      client_id;
    psa_handle_t handle;
    psa_handle_t status;
};

void mailbox_main(void)
{
    psa_signal_t   signals;
    psa_status_t   status;
    psa_msg_t      msg;
    struct client_param_t client_param;
    struct ___customized_t ns_msg;

    while (1) {
        signals = psa_wait(ALL, BLOCK);

        if (signals & MAILBOX_INTERRUPT_SIGNAL) {
            /* NS memory check needs to be performed. */
            ___customized_platform_get_mail(&ns_msg);

            /*
             * MACRO 'SID', 'VER', 'NSID', 'INVEC_LEN', 'OUTVEC_LEN', and
             * 'VECTORS' represent necessary information extraction from
             * 'ns_msg', put MACRO names here and leave the details to the
             * implementation.
             */
            if (ns_msg.type == PSA_IPC_CONNECT) {
                status = agent_psa_connect(SID(ns_msg), VER(ns_msg),
                                           NSID(ns_msg), &ns_msg);
            } else if (ns_msg.type == PSA_IPC_CLOSE) {
                status = agent_psa_close(ns_msg.handle, NSID(ns_msg));
            } else {
                /* Other types as call type and let API check errors. */
                client_param.ns_client_id_stateless = NSID(ns_msg);

                /*
                 * Use MACRO to demonstrate two cases: local vector
                 * descriptor and direct descriptor forwarding.
                 */

                /* Point to vector pointers in ns_msg. */
                PACK_VECTOR_POINTERS(client_param, ns_msg);
                status = agent_psa_call(ns_msg.handle,
                                        PARAM_PACK(ns_msg.type,
                                                   INVEC_LEN(ns_msg),
                                                   OUTVEC_LEN(ns_msg))|
                                                   NSIV | NSOV,
                                        &client_param,
                                        &ns_msg);
            }
```

(continues on next page)

(continued from previous page)

```
            /*
             * The service access reply is always fetched by a later
             * `psa_get` hence here only errors need to be dispatched.
             */
            error_dispatch(status);

        } else if (signals & ASYNC_MSG_REPLY) {
            /* The handle is freed for stateless service after 'psa_get'. */
            status       = psa_get(ASYNC_MSG_REPLY, &msg);
            ms_msg       = msg.rhandle;
            ns_msg.status = status;
            ___customized_platform___send_mail(&ns_msg);
        }
    }
}
```

Note: ___customized* API are implementation-specific APIs to be implemented by the mailbox Agent developer.

Customized manifest attribute

Three extra customized manifest attributes are added:

Name	Description
ns_agent	Indicate if manifest owner is an Agent.
client_id_base	The minimum client ID value (<0)
client_id_limit	The maximum client ID value (<0)

client_id_base and client_id_limit are negative numbers. This means that client_id_base <= client_id_limit, but abs(client_id_base) >= abs(client_id_limit). SPM can detect ID overlap when initialising secure partitions

The Non-secure callers are expected to provide a negative (<0) client ID when calling PSA API. A uniform mapping is implemented across all the NS agents, where the mapping is defined as the following:

NS client ID	Transformed client ID
-1	client_id_limit
-2	client_id_limit - 1
...	
-(abs(client_id_limit)-abs(client_id_base)+1)	client_id_base

Any other IDs provided by the NSPE will result in PSA_ERROR_INVALID_ARGUMENT.

Manifest tooling update

The manifest for agents involves specific keys ('ns_agent' e.g.), these keys give hints about how to achieve out-of-FF-M partitions which might be abused easily by developers, for example, claim partitions as agents. Some restrictions need to be applied in the manifest tool to limit the general secure service development referencing these keys.

Note: The limitations can mitigate the abuse but can't prevent it, as developers own all the source code they are working with.

One mechanism: adding a confirmation in the partition list file.

```
"description": "Non-Secure Mailbox Agent",
"manifest": "${CMAKE_SOURCE_DIR}/secure_fw/partitions/ns_agent_mailbox/ns_agent_mailbox.yaml
↪",
"non_ffm_attributes": "ns_agent", "other_option",
```

non_ffm_attributes tells the manifest tool that ns_agent is valid in ns_agent_mailbox.yaml. Otherwise, the manifest tool reports an error when a non-agent service abuses ns_agent in its manifest.

Runtime programming characteristics

Mailbox agent shall not be blocked by Agent-specific APIs. It can be blocked when:

- It is calling standard PSA Client APIs.
- It is calling psa_wait().

IDLE processing

Only ONE place is recommended to enter IDLE. The place is decided based on the system topologies:

- If there is one Trustzone-based NSPE, this NSPE is the recommended place no matter how many mailbox agents there are in the system.
- If there are only mailbox-based NSPEs, entering IDLE can happen in one of the mailbox agents.

The solution is:

- An IDLE entering API is provided in SPRTL.
- A partition without specific flag can't call this API.
- The manifest tooling counts the partitions with this specific flag, and assert errors when multiple instances are found.

Copyright (c) 2022-2024, Arm Limited. All rights reserved. Copyright (c) 2023 Cypress Semiconductor Corporation (an Infineon company) or an affiliate of Cypress Semiconductor Corporation. All rights reserved.

Memory Access Check of Trusted Firmware-M in Multi-Core Topology

Author
David Hu

Organization
Arm Limited

Contact
david.hu@arm.com

Introduction

TF-M memory access check function tfm_has_access_to_region() checks whether an access has proper permission to read or write the target memory region.

On single Armv8-M core platform based on Trustzone, TF-M memory access check implementation relies on Armv8-M Security Extension (CMSE) intrinsic cmse_check_address_range(). The secure core may not implement CMSE on multi-core platforms. Even if CMSE is implemented on a multi-core platform, additional check on system-level security and memory access management units are still necessary since CMSE intrinsics and TT instructions are only aware of MPU/SAU/IDAU inside the secure core.

As a result, TF-M in multi-core topology requires a dedicated access check process which can work without CMSE support. This document discuss about the design of the memory access check in multi-core topology.

Overall Design

Memory Access Check Policy

The policies vary in diverse Isolation Levels.

When TF-M works in Isolation Level 1, the access check in multi-core topology checks

- Memory region is valid according to system settings
- Non-secure client call request should not access secure memory.
- Secure services should not directly access non-secure memory. According to PSA Firmware Framework, Secure services should call Secure Partition APIs to ask TF-M SPM to fetch non-secure input/output buffer for them.
- Whether read/write attribute match between access and memory region

When TF-M works in Isolation Level 2, the access check in multi-core topology checks:

- Memory region is valid according to system settings
- Non-secure client call request should not access secure memory.
- Secure services should not directly access non-secure memory. According to PSA Firmware Framework, Secure services should call Secure Partition APIs to ask TF-M SPM to fetch non-secure input/outputs buffer for them.
- Whether read/write attribute match between access and memory region
- Unprivileged secure access should not access privileged secure memory region

The check policy in Isolation Level 3 will be defined according to TF-M future implementation.

The above policies will be adjusted according to TF-M implementation and PSA specs.

General Check Process in TF-M Core

In multi-core topology, tfm_has_access_to_region() is still invoked to keep an uniform interface to TF-M SPM. The function implementation should be placed in multi-core topology specific files separated from Trustzone-based access check.

Multi-core platform specific tfm_hal_memory_check() can invoke tfm_has_access_to_region() to implement the entire memory access check routine.

During the check process, tfm_has_access_to_region() compares the access permission with memory region attributes and determines whether the access is allowed to access the region according to policy described in *Memory Access Check Policy* above.

tfm_has_access_to_region() invokes 3 HAL APIs to retrieve attributes of target memory region.

- tfm_hal_get_mem_security_attr() retrieves the security attributes of the target memory region.
- tfm_hal_get_secure_access_attr() retrieves secure access attributes of the target memory region.
- tfm_hal_get_ns_access_attr() retrieves non-secure access attributes of the target memory region.

All three functions are implemented by multi-core platform support. The definitions are specified in the section *HAL APIs* below.

The pseudo code of tfm_has_access_to_region() is shown below.

```
int32_t tfm_has_access_to_region(const void *p, size_t s, uint8_t flags)
{
    struct security_attr_info_t security_attr;
    struct mem_attr_info_t mem_attr;

    /* Validate input parameters */
    if (Validation failed) {
        return error;
    }

    /* The memory access check should be executed inside TF-M PSA RoT */
    if (Not in privileged level) {
        abort;
    }

    /* Set initial value */
    security_attr_init(&security_attr);

    /* Retrieve security attributes of memory region */
    tfm_hal_get_mem_security_attr(p, s, &security_attr);

    /* Compare according to current Isolation Level */
    if (Input flags mismatch security attributes) {
        return error;
    }

    /* Set initial value */
    mem_attr_init(&mem_attr);

    if (The target memory region is in secure memory space) {
        /* Retrieve access attributes of secure memory region */
        tfm_hal_get_secure_access_attr(p, s, &mem_attr);

        if (Not in Isolation Level 1) {
            /* Secure memory protection unit(s) must be enabled in Isolation Level 2 and 3 */
            if (Protection unit not enabled) {
                abort;
            }
        }
    } else {
        /* Retrieve access attributes of non-secure memory region. */
        tfm_hal_get_ns_access_attr(p, s, &mem_attr);
    }

    /* Compare according to current Isolation Level and non-secure/secure access. */
    if (Input flags match memory attributes) {
        return success;
    }

    return error;
}
```

Note: It cannot be guaranteed that TF-M provides a comprehensive memory access check on non-secure memory for NSPE client call. If non-secure memory protection or isolation is required in a multi-core system, NSPE software should implement and execute the check functionalities in NSPE, rather than relying on TF-M access check.

For example, all the access from NSPE client calls to non-secure memory are classified as unprivileged in current TF-M implementation. Multi-core access check may skip the privileged/unprivileged permission check for non-secure access.

If a multi-core system enforces the privileged/unprivileged isolation and protection of non-

secure area, NSPE software should execute the corresponding check functionalities before submitting the NSPE client call request to SPE.

Data Types and APIs

Data Types

Access Permission Flags

The following flags are defined to indicate the access permission attributes. Each flag is mapped to the corresponding CMSE macro. Please refer to ARMv8-M Security Extensions: Requirements on Development Tools for details of each CMSE macro.

MEM_CHECK_MPU_READWRITE

Mapped to CMSE macro CMSE_MPU_READWRITE to indicate that the access requires both read and write permission to the target memory region.

```
#define MEM_CHECK_MPU_READWRITE   (1 << 0x0)
```

MEM_CHECK_MPU_UNPRIV

Mapped to CMSE macro CMSE_MPU_UNPRIV to indicate that it is an unprivileged access.

```
#define MEM_CHECK_MPU_UNPRIV      (1 << 0x2)
```

MEM_CHECK_MPU_READ

Mapped to CMSE macro CMSE_MPU_READ. It indicates that it is a read-only access to target memory region.

```
#define MEM_CHECK_MPU_READ        (1 << 0x3)
```

MEM_CHECK_NONSECURE

Mapped to CSME macro CMSE_NONSECURE to indicate that it is a access from non-secure client call request. If this flag is unset, it indicates the access is required from SPE.

```
#define MEM_CHECK_AU_NONSECURE    (1 << 0x1)
#define MEM_CHECK_MPU_NONSECURE   (1 << 0x4)
#define MEM_CHECK_NONSECURE       (MEM_CHECK_AU_NONSECURE | \
                                   MEM_CHECK_MPU_NONSECURE)
```

Security Attributes Information

The structure security_attr_info_t contains the security attributes information of the target memory region. tfm_hal_get_mem_security_attr() implementation should fill the structure fields according to the platform specific secure isolation setting.

```
struct security_attr_info_t {
    bool is_valid;
    bool is_secure;
};
```

is_valid indicates whether the target memory region is valid according to platform resource assignment and security isolation configurations.

is_secure indicates the target memory region is secure or non-secure. The value is only valid when is_valid is true.

Memory Attributes Information

The structure mem_attr_info_t contains the memory access attributes information of the target memory region. tfm_hal_get_secure_access_attr() and tfm_hal_get_ns_access_attr() implementations should fill the structure fields according to the memory protection settings.

```
struct mem_attr_info_t {
    bool is_mpu_enabled;
    bool is_valid;
    bool is_xn;
    bool is_priv_rd_allow;
    bool is_priv_wr_allow;
    bool is_unpriv_rd_allow;
    bool is_unpriv_wr_allow;
};
```

is_mpu_enabled indicates whether the MPU and other management unit are enabled and work normally.
is_valid indicates whether the target memory region is valid according to platform resource assignment and memory protection configurations.
is_xn indicates whether the target memory region is Execute Never. This field is only valid when is_valid is true.
is_priv_rd_allow and is_priv_wr_allow indicates whether the target memory region allows privileged read/write. Both the fields are valid only when is_valid is true.
is_unpriv_rd_allow and is_unpriv_wr_allow indicates whether the target memory region allows unprivileged read/write. Both the fields are valid only when is_valid is true.

HAL APIs

tfm_hal_get_mem_security_attr()

tfm_hal_get_mem_security_attr() retrieves the current active security configuration information and fills the security_attr_info_t.

```
void tfm_hal_get_mem_security_attr(const void *p, size_t s,
                                   struct security_attr_info_t *p_attr);
```

Parameters	
p	Base address of the target memory region
s	Size of the target memory region
p_attr	Pointer to the security_attr_info_t to be filled
Return	
void	None

The implementation should be decoupled from TF-M current isolation level or access check policy.

All the fields in security_attr_info_t shall be explicitly set in tfm_hal_get_mem_security_attr().

If the target memory region crosses boundaries of different security regions or levels in security isolation configuration, tfm_hal_get_mem_security_attr() should determine whether the memory region violates current security isolation. It is recommended to mark the target memory

region as invalid in such case, even if the adjoining regions or levels have the same security configuration.

If the target memory region is not explicitly specified in memory security configuration, tfm_hal_get_mem_security_attr() can return the following values according to actual use case:

- Either set is_valid = false
- Or set is_valid = true and set is_secure according to platform specific policy.

tfm_hal_get_secure_access_attr()

tfm_hal_get_secure_access_attr() retrieves the secure memory protection configuration information and fills the mem_attr_info_t.

```
void tfm_hal_get_secure_access_attr(const void *p, size_t s,
                                    struct mem_attr_info_t *p_attr);
```

Parameters	
p	Base address of the target memory region
s	Size of the target memory region
p_attr	Pointer to the mem_attr_info_t to be filled
Return	
void	None

The implementation should be decoupled from TF-M current isolation level or access check policy.

All the fields in mem_attr_info_t shall be explicitly set in tfm_hal_get_secure_access_attr(), according to current active memory protection configuration. It is recommended to retrieve the attributes from secure MPU and other hardware memory protection unit(s). The implementation can also be simplified by checking static system-level memory layout.

If the target memory region is not specified in current active secure memory protection configuration, tfm_hal_get_secure_access_attr() can select the following values according to actual use case.

- Either directly set is_valid to false
- Or set is_valid to true and set other fields according to other memory assignment information, such as static system-level memory layout.

If secure memory protection unit(s) is *disabled* and the target memory region is a valid area according to platform resource assignment, tfm_hal_get_secure_access_attr() must set is_mpu_enabled to false and set other fields according to current system-level memory layout.

tfm_hal_get_ns_access_attr()

tfm_hal_get_ns_access_attr() retrieves the non-secure memory protection configuration information and fills the mem_attr_info_t.

```
void tfm_hal_get_ns_access_attr(const void *p, size_t s,
                                struct mem_attr_info_t *p_attr);
```

Parameters	
p	Base address of the target memory region
s	Size of the target memory region
p_attr	Pointer to the mem_attr_info_t to be filled
Return	
void	None

The implementation should be decoupled from TF-M current isolation level or access check policy.

Since non-secure core runs asynchronously, the non-secure MPU setting may be modified by NSPE OS and therefore the attributes of the target memory region can be unavailable during tfm_hal_get_ns_access_attr() execution in TF-M. When the target memory region is not specified in non-secure MPU, tfm_hal_get_ns_access_attr() can set the fields according to other memory setting information, such as static system-level memory layout.

If non-secure memory protection unit(s) is *disabled* and the target memory region is a valid area according to platform resource assignment, tfm_hal_get_ns_access_attr() can set the following fields in mem_attr_info_t to default values:

- is_mpu_enabled = false
- is_valid = true
- is_xn = true
- is_priv_rd_allow = true
- is_unpriv_rd_allow = true

is_priv_wr_allow and is_unpriv_wr_allow can be set according to current system-level memory layout, such as whether it is in code section or data section.

General retrieval functions

TF-M implements 3 general retrieval functions to retrieve memory region security attributes or memory protection configurations, based on static system-level memory layout. Platform specific HAL functions can invoke those 3 general functions to simplify implementations.

- tfm_get_mem_region_security_attr() retrieves general security attributes from static system-level memory layout.
- tfm_get_secure_mem_region_attr() retrieves general secure memory protection configurations from static system-level memory layout.
- tfm_get_ns_mem_region_attr() retrieves general non-secure memory protection configurations from static system-level memory layout.

If a multi-core platform's memory layout may vary in runtime, it shall not rely on these 3 functions to retrieve static configurations. These 3 functions run through memory layout table to check against each memory section one by one, with pure software implementation. It might cost more time compared to hardware-based memory access check.

tfm_get_mem_region_security_attr()

tfm_get_mem_region_security_attr() retrieves security attributes of target memory region according to the static system-level memory layout and fills the security_attr_info_t.

```
void tfm_get_mem_region_security_attr(const void *p, size_t s,
                                      struct security_attr_info_t *p_attr);
```

Parameters	
p	Base address of the target memory region
s	Size of the target memory region
p_attr	Pointer to the security_attr_info_t to be filled
Return	
void	None

tfm_get_secure_mem_region_attr()

tfm_get_secure_mem_region_attr() retrieves general secure memory protection configuration information of the target memory region according to the static system-level memory layout and fills the mem_attr_info_t.

```
void tfm_get_secure_mem_region_attr(const void *p, size_t s,
                                    struct mem_attr_info_t *p_attr);
```

Parameters	
p	Base address of the target memory region
s	Size of the target memory region
p_attr	Pointer to the mem_attr_info_t to be filled
Return	
void	None

tfm_get_ns_mem_region_attr()

tfm_get_ns_mem_region_attr() retrieves general non-secure memory protection configuration information of the target memory region according to the static system-level memory layout and fills the mem_attr_info_t.

```
void tfm_get_ns_mem_region_attr(const void *p, size_t s,
                                struct mem_attr_info_t *p_attr);
```

Parameters	
p	Base address of the target memory region
s	Size of the target memory region
p_attr	Pointer to the mem_attr_info_t to be filled
Return	
void	None

Copyright (c) 2019-2023, Arm Limited. All rights reserved.

Copyright (c) 2020, Arm Limited. All rights reserved.

Secure Services

Secure Partition Manager

This document describes the Secure Partition Manager (*SPM*) implementation design in Trusted Firmware-M (*TF-M*).

Note:

- The FF-M in this document refers to the accumulated result of two specifications: FF-M v1.1 Update on FF-M v1.0.
- The words marked as *interpreted* are defined terms. Find the terms in referenced documents if it is not described in this document.

Introduction

The service access process of FF-M:

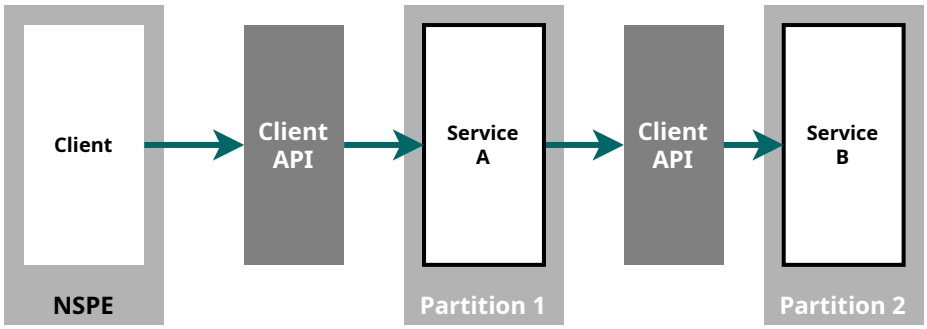


Fig. 5: FF-M service access process

Secure services (aka *Service*) is the component providing secure functionalities in *SPE*, and *Client* is the user of the *Service*. A service acts as a client when it is accessing its depending services.

Services are grouped into *Secure Partition* (aka *partition*). A partition:

- Contains services with the same purpose.
- Provides implementation required isolation boundaries.
- Is a software development unit.

Each service exposes its *Service ID* (*SID*) and *Handle* for client access usage. Clients access services by *SID* or *Handle* through FF-M *Client API*. Partitions use FF-M *Secure Partition API* when it needs to operate on client data or reply to a client.

SPM is the centre of an FF-M compliant implementation, which sets up and maintains a firmware framework that:

- Implements *Client API* and *Secure Partition API*.
- Manages partition runtime to follow FF-M.
- Involves necessary implementation-defined items to support the implementation.

SPM interfaces consist of these two categories:

- FF-M defined API.
- Extended API to support the implementation.

Both API categories are compliant with FF-M concepts and guidelines. The core concept of TF-M SPM surrounds the FF-M defined service management and access process. Besides this, another important implementation part is partition runtime management.

Partition runtime model

One partition must work under as *ONE* of the runtime models: *Inter-process communication* (*IPC*) model or *Secure Function* (*SFN*) model.

A partition that runs under the *IPC* model looks like a classic *process*. There is *ONE* thread inside the partition keeps waiting for signals. SPM converts the service accessing info from the *Client API* call into messages and assert a signal to the partition. The partition calls corresponded service function indicated by the signal and its bound message, and reply service returned result to the client. The advantages of this model:

- It provides better isolation by limiting the interfaces on data interactive. Data are preferred to be processed in a local buffer.
- It provides a mechanism for handling multiple service access. There is no memory mapping mechanism in the MCU system, hence it is hard to provide multiple function call contexts

when serving multiple-threaded clients if the service access is implemented in a function-call based mechanism. This model converts multiple service accesses into messages, let the partition handles the service access in messages one by one.

The *Secure Function* (*SFN*) model partition is close to a *library*. Each service is provided as a function entry inside the partition. SPM launches the target service function after the service is found. The whole procedure (from client to service function) is a function call. This model:

- Saves the workloads spent on *IPC* scheduling.

Meanwhile, it relaxes the data interactive mechanism, for example, allow direct memory access (MMIOVEC). And it is hard to enable multiple-threaded clients service access because of multiple call context-maintenance difficulties.

An implementation contains only *SFN* partitions fits better in the resource-constrained devices, it is called an *SFN model implementation*. And it is an *IPC model implementation* when *IPC* partitions exist in the system.

Note: *IPC model implementation* can handle access to the services in the *SFN* partition.

Components and isolation levels

There are *THREE* isolation levels defined in *FF-M*. These levels can fulfil different security requirements by defining different isolation boundaries.

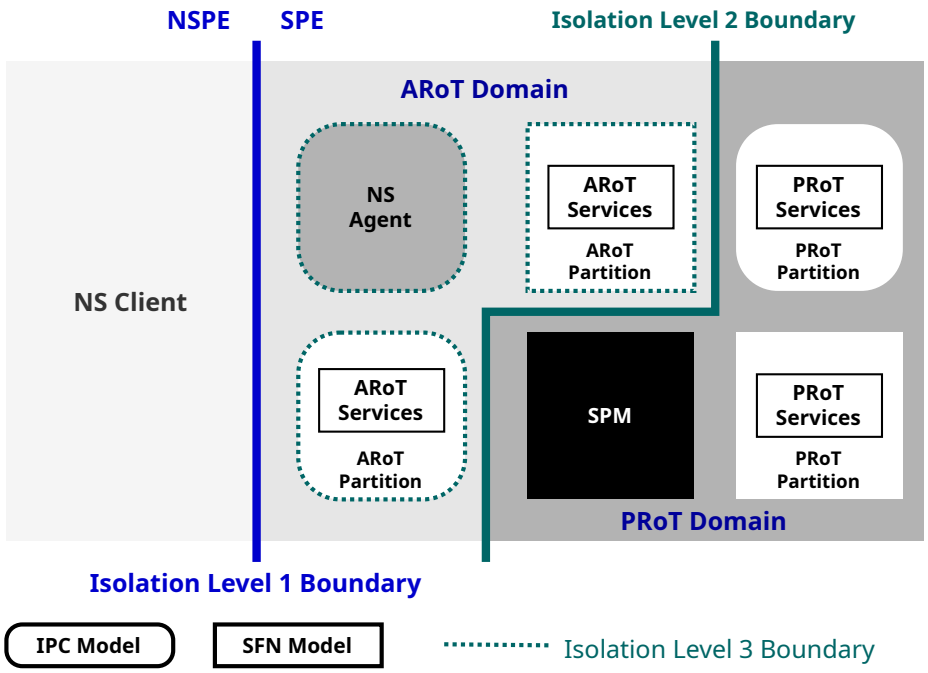


Fig. 6: Components and isolation boundaries

Note: Concept *ARoT*, *PRoT*, *domain*, and boundaries are in the *FF-M* specification.

Not like an *SPE* client that can call *Client API* to access the secure services in one step, an *NSPE* client needs to cross the secure boundaries first before calling *Client API*. The component *NS Agent* in fig-modelisolation represents *NSPE* clients after they crossed the secure boundaries. This could help *SPM* handles the request in a unified way instead of care about the special boundaries.

Note: *NS Agent* is a necessary implementation-defined component out of FF-M specification. *NS Agent* has a dedicated stack because secure and non-secure can not share the stack. It also has dedicated execution bodies. For example, RPC-based *NS Agent* has a while loop that keeps waiting for messages; and Trustzone-based *NS Agent* has veneer code to take over *NSPE* secure call. This makes *NS Agent* to be a component more like a *process*. Hence in the simplest implementation (*SFN model implementation* mentioned above), *NS Agent* is the only process in the system, the scheduling logic can be extremely simplified since no other process execution needs to be scheduled. But the scheduling interface is still necessary to SPM, this could help SPM treat both *SFN* and *IPC* model implementation in a unified way.

Check *NS Agent* for details.

Implementation principle

The principles for TF-M SPM implementation:

Important:

- SPM can treat these components as the client: NS Agent, SFN Partition, and IPC partition.
- These components can provide services: SFN Partition, IPC partition, and built-in services. A built-in service is built up with SPM together.
- All partition services must be accessed by *Client API*.
- Partitions interact with client data by *Secure Partition API*.
- Built-in services are strongly recommended to be accessed by *Client API*. Customized interfaces are restricted.
- Built-in services can call SPM internal interfaces directly.

Runtime management

The runtime execution runs among the components, there are **4** runtime states:

- *Initializing* state, to set up the SPM runtime environment after system powers up
- *IDLE* state, when SPM runtime environment is set up and partitions are ready for service access.
- *Serving* state, when partition is under initializing or service access handling.
- *Background* state, such as the arrival of secure interrupt or unexpected faults. *Background* state returns to the state it preempts. *Background* state can be nested.

The state transition diagram:

Initializing

The goal of TF-M initializing is to perform necessary initialization and move to the *Serving* state. This state starts with platform-specific power on sequence, then *SPM* takes over the execution to perform these operations:

1. A preparation initialization process before SPM runtime initialization.
2. SPM runtime initialization.
3. A post initialization happens after the SPM runtime initialization and before the first partition gets launched.

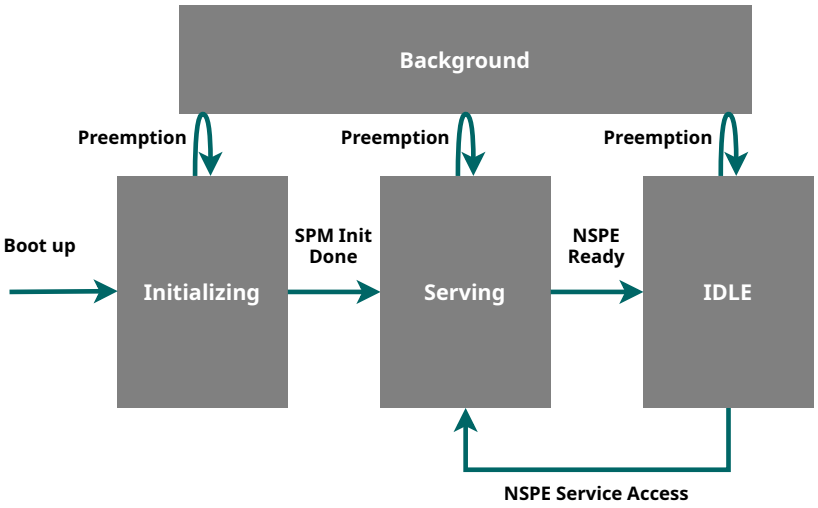


Fig. 7: SPE runtime execution states

Note: These procedures and their sub-routines are recommended to be applied with execution measurement mechansim to mitigate the *Hardware Fault Injection* attack.

Preparation initialization

The purpose of this preparation initialization is to provide a chance for performing those security required but generic platform power-on skipped operations, such as:

- Restrict *SPM* execution, for example, set up memory overflow settings for SPM runtime memory, or set code out of SPM as un-executable, even though SPM is a privileged component in general.

Note: The logging-related peripheral can be set up **AT THIS STEP**, if logging is enabled and it needs peripheral support. There is no standalone initializing HAL API proposed for logging, so here is an ideal place for initializing them.

This procedure is abstracted into one *HAL*, and a few example procedures are implemented as its sub-routines for reference:

- Architecture extensions initialization, Check chapter *Architecture security settings* for detailed information.
- Isolation and lifecycle initialization.

The load isolation boundaries need to be set up here, such as SPE/NSPE boundary, and ARoT/PRoT boundary if isolation level 2 is applied.

The lifecycle is initiated by a secure bootloader usually. And in this stage of SPM initializing, SPM double-checks the lifecycle set up status (following a specific lifecycle management guidelines). Note that the hardware debugger settings can be part of lifecycle settings.

Important: Double-check debugger settings when performing a product release.

SPM runtime initialization

This procedure initializes necessary runtime operations such as memory allocator, loading partitions and partition-specific initialization (binding partitions with platform resources).

The general processes:

1. Initialize runtime functionalities, such as memory allocator.
2. Load partitions by repeating below steps:
    - Find a partition load information.
    - Allocate runtime objects for this partition.
    - Link the runtime objects with load information.
    - Init partition contexts (Thread and call context).
    - Init partition isolation boundaries (MMIO e.g.).
    - Init partition interrupts.

After no more partitions need to be loaded, the SPM runtime is set up but partitions' initialization routines have not run yet - the partition runtime context is initialized for the routine call.

The partition initialization routine is a special service that serves SPM only, because:

- SPM needs to call the initialization routine, just like it calls into the service routine.
- The partition initialization routine can access its depending services. Putting initialization routine in the same runtime environment as common service routines can avoid special operations.

Hence a *Partition initialization client* needs to be created to initialize the SFN partitions, because:

- *SPM runtime initialization* happen inside a special runtime environment compare to the partition runtime execution, then an environment switching is needed.
- IPC partitions are initialized by the scheduler and dependencies are handled by signals and messages asynchronously, hence IPC partitions can process the dependencies by their own.

The *Partition initialization client* is created differently based on the implementation runtime model:

- A SFN client is created under the SFN model implementation.
- A IPC client is created under the IPC model implementation. This client thread has the highest priority.

As the other partitions, the client is created with context standby, and it is executed after the *Post initialization* stage.

Post initialization

Platform code can change specific partition settings in this procedure before partitions start. A few SPM API is callable at this stage, such as set a signal into a specific partition, or customized peripheral settings.

Serving

Two execution categories work under this state:

- *Partition initialization routine execution*.
- *Secure service access*.

This state indicates the serving is ongoing. It is mainly the service routine execution, plus a few SPM executions when SPM API gets called.

Important: The service access process introduced in this chapter (Such as *Secure service access*) is abstracted from the FF-M specification. Reference the FF-M specification for the details of each step.

Partition initialization routine execution

The partition initialization routines get called. One partition may access its depending services during initializing, then this procedure is a *Secure service access*.

The initialization routine gets called initially by *Partition initialization client*, also can be called by Client API before service access, if the target partition is not initialized but a service access request is raised by one client.

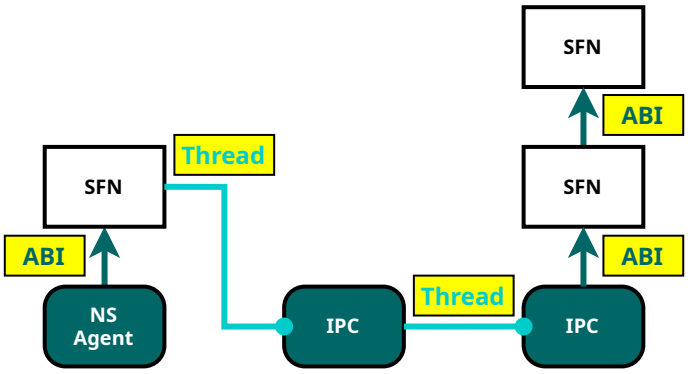
Secure service access

The process of service access:

1. A *client* calls an FF-M Client API.
2. *SPM* validates inputs and looks up for the targeted service.
3. *SPM* constructs the request to be delivered under a proper runtime mechanism.
4. The target service gets executed. It can perform internal executions or access depending services to prepare the response. It also can wait for specific signals.
5. The target service calls FF-M Secure Partition API to request a reply to the client.
6. SPM delivers the response to the client, and the API called by the client returns.

The mechanism of how SPM interact with the target partition depends on the partition runtime model.

- Access to a service in an SFN partition is a function call, which does not switch the current process indicator.
- Access to a service in an IPC partition leads to scheduling, which switches the current process indicator.
- When the execution roams between components because of a function call or scheduling, the isolation boundaries NEED to be switched if there are boundaries between components.



No matter what kind of partition a client is trying to access, the SPM API is called firstly as it is the interface for service access. There are two ABI types when calling SPM API: Cross-boundary or No-cross-boundary.

Calling SPM API

SPM is placed in the PRoT domain. It MAY have isolation boundaries under particular isolation levels. For example:

- There are boundaries between ARoT components and SPM under isolation level 2 and 3.

The API SPM provided needs to support the function call (no boundary switching) and cross-boundary call. A direct call reaches the API entrance directly, while a cross-boundary call needs a mechanism (Supervisor call e.g.) to cross the boundary first before reaching the API entrance.

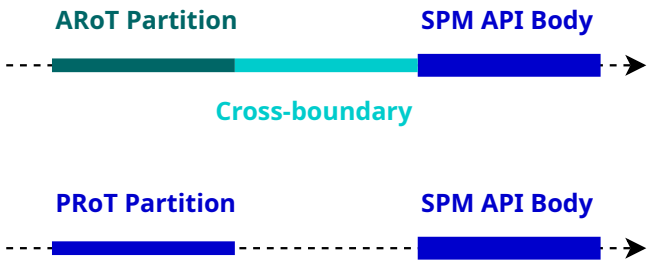


Fig. 8: SPM call types

SPM internal execution flow

SPM internal execution flow as shown in diagram:

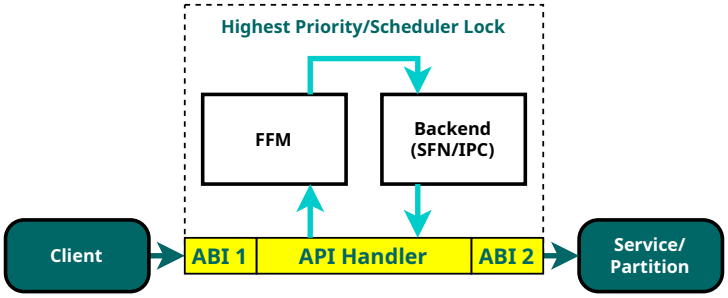


Fig. 9: SPM API runtime

The process:

- PSA API gets called by one of the ABI mentioned in the last chapter as *ABI 1* in the diagram.
- The unified API Handler calls FF-M and backend subroutines in sequence.
- The *FF-M* subroutine performs *FF-M* defined operations.
- The backend operations perform target partition runtime model decided operations. For example, enqueue message into the target partition under the IPC runtime model, or prepare to call context with the message as the parameters under the SFN runtime model.
- API Handler triggers different ABI based on the result of the backends.

The API handler:

- Can process the *PROGRAMMER_ERROR* in a unified place.
- Can see the prepared caller and callee context, with exited SPM context. It is an ideal place for subsequent operations such as context switching.

An example code:

```
void abi(void *p)
{
    status = spm_api(p);
    /*
     * Now both the caller and callee contexts are
     * managed by spm_api.
     */
    if (status == ACTION1) {
        /*
         * Check if extra operations are required
         * instead of a direct return.
         */
        exit_action1();
    }
}
```

The explanation about *Scheduler Lock*:

Some FF-M API runs as a generic thread to prevent long time exclusive execution. When a preemption happens, a new partition thread can call SPM API again, makes SPM API nested. It needs extra memory in SPM to be allocated to store the preempted context. Lock the scheduler while SPM API is executing can ensure SPM API complete execution after preemption is handled. There can be multiple ways to lock the scheduler:

- Set a scheduler lock.
- Set SPM API thread priority as the highest.

Backend service messaging

A message to service is created after the target service is found and the target partition runtime model is known. The preparation before ABI triggers the final accessing:

- The message is pushed into partition memory under a specific ABI mechanism if the target partition model is *SFN* and there are boundaries between SPM and the target partition. After this, requests a specific call type to the SPM ABI module.
- The target service routine is called with the message parameter if there are no boundaries between SPM and the target partition and the partition runtime is *SFN*.
- The message is queued into the partition message list if the target partition runtime model is *IPC*.
- IPC partition replies to the client by *psa_reply*, which is another SPM API call procedure.
- SFN partition return triggers an implied *psa_reply*, which is also another SPM API call procedure.

Note: The backends also handle the isolation boundary switching.

Sessions and contexts

FF-M API allows multiple sessions for a service if the service is classic connection-based. The service can maintain multiple local session data and use *rhandle* in the message body to identify which client this session is bound with.

But this does not mean when an ongoing service accessing is preempted, another service access request can get a chance for new access. This is because of the limited context storage - supporting multiple contexts in a common service costs much memory, and runtime operations (allocation and re-location). Limited the context content in the stack only can mitigate the effort, but this requirement requires too much for the service development.

The implementation-decisions are:

- IPC partitions handles messages one by one, the client gets blocked before the service replying to the client.
- The client is blocked when accessing services are handling a service request in an SFN partition.

ABI type summary

The interface type is decided by the runtime model of the target component. Hence PSA API has two types of ABI: *Cross-boundary ABI* and *Function call ABI*. After SPM operations, one more component runtime type shows up: The IPC partition, hence *schedule* is the mechanism when accessing services inside an IPC partition.

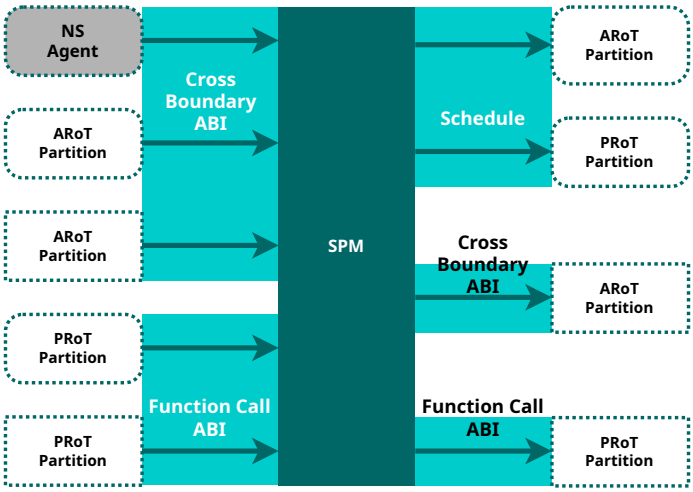


Fig. 10: ABI types

Note: The API that does not switch context returns directly, which is not covered in the above diagram.

IDLE state

The *IDLE state* can be represented by the *NS Agent* action:

- Launching NSPE software (Trustzone case, e.g.), or send a signal to NSPE software (RPC case, e.g.).

It is because *NS Agent* is the last component being initialized in the system. Its execution indicates other partitions' initialization has accomplished.

Background state

Background execution can happen at any time when the arrival of interrupts or execution faults. An ongoing background execution indicates the state is a *Background state*. The characteristics:

- The background state has a higher execution priority than other states - other states stall when the background state is executing.
- Background execution can be nested. For example, an interrupt handler can preempt an ongoing interrupt execution.
- Particular partition code can be involved in the background state, for example, the *First Level Interrupt Handler (FLIH)* of one partition.
- Background state MUST return to the state it preempts.

Note: Interrupt handling is a common background state example. Check Interrupt design document for details.

Practical implementation items

This chapter describes the practical implementation contents.

Important: Arm M-profile architecture is the default hardware architecture when describing architecture-specific items.

The general M-profile programming is not involved in this document. The following chapters introduce the mandatory settings for security requirements.

Architecture security settings

When an *Armv8m Security Extension* (Aka *Trustzone-M*) is available in the system, these settings are required to be set:

- The MSPLIM needs to be set correctly to prevent stack overflow.
- The exception handler priority needs to be decided.
- Boost the secure handler mode priority to prevent NSPE from preempting SPE handler mode execution(*AIRCR.PRIS*).
- Disable NSPE hardware faults when a secure fault is happening. Trap in the secure fault with the highest priority can be a valid option.
- Push seals on the stack top when a stack is allocated (*TFMV-1*). Also check *Stack seal* chapter for details.

Besides *Armv8m Security Extension*, these settings need to care when *Floatpoint Extension* is enabled for partition usage:

- *FPCCR.TS*, *FPCCR.CLRONRET* and *FPCCR.CLRONRETS* need to be set when booting.
- *CPACR.CP10* and *CPACR.CP11* need to be set when booting.

Important: Floatpoint usage is prohibited in SPM and background execution.

Stack seal

When Trustzone-M is applied, the architecture specification recommends sealing the secure stack by:

- Push two *SEAL* values (*0xFEF5EDA5*) at the stack bottom, when a stack is allocated.
- Push two *SEAL* values on the stack pointer which is going to be switched out.

Check architecture specification and vulnerability *TFMV-1* for details.

Trustzone-M reentrant

The Trustzone-M has characteristics that:

- SPE keeps the last assigned stack pointer value when execution leaves SPE.
- SPE execution can be preempted by NSPE which causes an execution left.

It is possible that NSPE preemption caused a second thread calls into SPE and re-uses the secure stack contains the first thread's context, which obviously causes information leakage and runtime state inconsistent.

Armv8.1-M provides the hardware setting *CCR_S.TRD* to prevent the reentrant. On an Armv8.0-M architecture, extra software logic needs to be added at the veneer entry:

- Check if the local stack points to a *SEAL* when veneer code get executed.

```
/* This is a theoretical code that is not in a real project. */
veneer() {
    content = get_sp_value();
    if (context != SEAL) /* Error if reentrant detected */
        error();
}
```

SPM Runtime ABI

This chapter describes the runtime implementation of SPM.

Scheduling

The scheduling logic is put inside the PendSV mode. PendSV mode's priority is set as one level higher than the default thread mode priority. If *Trustzone-M* is present, the priority is set as the lowest just above NS exception priority to prevent a preemption in secure exceptions.

PendSV is an ideal place for scheduling logic, because:

- An interrupt triggered scheduling during PendSV execution lead to another PendSV execution before exception return to the thread mode, which can find the latest run-able thread.

Function call ABI

In the diagram fig-abi_scheduler, the ABI can have two basic types: cross-boundary and direct call (No-cross-boundary).

When applying *SVCall* (*SVC*) as the cross-boundary mechanism, the implementation can be straight like:

- The SVC handler calls SPM internal routines, and eventually back to the handler before an exit.

Under the IPC model implementation, to re-use *ABI 2* in *No-cross-boundary*, a software ABI needs to be provided.

While under the SFN model plus isolation level 1, both *ABI 1* and *ABI 2* can be a direct function call.

NS Agent

The *NS Agent* (*NSA*) forwards NSPE service access request to SPM. It is a special *partition* that:

- It does not provide FF-M aligned secure services.
- It runs with the second-lowest priority under *IPC model implementation* (The IDLE thread has the lowest priority).
- It has isolation boundaries and an individual stack.
- It requires specific services and mechanisms compared to common partitions.

There are two known types for NS Agent:

- Trustzone-M based.
- Remote Procedure Call (RPC) based.

This process is put inside the ARoT domain, to prevent assign unnecessary PRoT permissions to the NSPE request parsing logic.

Trustzone-M specific

The functionalities of a Trustzone-M specific NSA is:

- Launch NSPE when booting.
- Wait in the veneer code, and get executed when NSPE accesses services.

As there may be multiple NSPE threads calling into SPE, and SPM wants to identify them, special mechanisms can be proposed to provide the identification. Check specific NS ID client ID or context related documents for details.

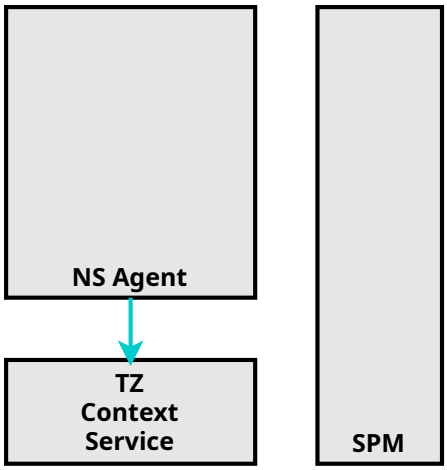


Fig. 11: TZ NSA and specific service

RPC specific

Compared to Trustzone-M NSA, RPC NSA looks closer to a generic partition:

- It has a message loop, keep waiting for RPC events.
- It converts received RPC events into FF-M API call to target services.

And compared to generic partitions, the differences are:

- It parses RPC messages to know which NSPE thread is accessing services. Hence it needs special interfaces to help SPM to identify the NSPE clients.
- It needs to check NSPE client memory and map to local before calling SPM API.
- It cannot be blocked during API calls, which affects handling the RPC requests.

Partition

A partition is a set of services in the same scope. Services are generally implemented as functions, and the partition exposes the services in different ways based on the partition model: *IPC* or *SFN*.

A partition build generates these outputs:

- A partition load information, used by SPM.
- A partition program containing service interface and logic, typically a library.
- An optional service API set for easier client usage, by encapsulating the low-level *FF-M* Client API. These API needs to be integrated into client space.

Partition loading

SPM needs to set up runtime objects to manage partitions by parsing the load information of partitions. In general, the partition load information is stored in a const memory area can be random read directly, hence SPM can direct link runtime objects to the load information without a copy operation. This is called a *Static Load* mechanism.

Each partition has different numbers of dependencies and services, this makes the load information size of each partition different, it would be hard to put such variable size elements in an array. The solution here is putting these elements in a dedicated section, for SPM enumerating while loading. Each partition can define variable size load information type based on the common load info type.

The common load information:

```
struct partition_load_info_t {
    uint32_t        psa_ff_ver;     /* Encode the version with magic    */
    int32_t         pid;            /* Partition ID                     */
    uint32_t        flags;          /* ARoT/PRoT, SFN/IPC, priority     */
    uintptr_t       entry;          /* Entry point                      */
    size_t          stack_size;     /* Stack size                       */
    size_t          heap_size;      /* Heap size                        */
    uint32_t        ndeps;          /* Dependency number                */
    uint32_t        nservices;      /* Service number                   */
    uint32_t        nassets;        /* Asset numbers                    */
    uint32_t        nirqs;          /* Number of IRQ owned by Partition */
};

And the example for a specific partition load info:
struct partition_example_load_info_t {
    struct partition_load_info_t ldi;      /* Common info info         */
    uint32_t                     deps[10]; /* Dependencies             */
    /*                 ... other infos ...                             */
};
```

Peripheral binding

A partition can declare multiple peripherals (Interrupts are part of peripherals). The peripherals binding process:

- The tooling references symbols in a fixed pattern in the partition load information.
- The HAL implementation needs to provide the symbols being referenced.
- SPM calls HAL API to bind the partition info with devices when the partition gets loaded.
- The platform HAL acknowledges the binding if validation pass on SPM given load information.

Integration and development

These modules are expected to be object/library level modularised, each module should be generated into object/library at build time:

Table 40: Object level modularization

Name	Description
SPM	All SPM related modules such as SPM, system, and so on.
Platform	Platform sources are switchable.
Services and Secure Partition	These items should be standalone.
Service Runtime Library	This is a shared runtime library.

HAL

The HAL here mainly refers to the SPM HAL. The SPM HAL implementation is running with the same privilege level and hardware mode with SPM. The implementation is object level modularized with SPM.

Check the *HAL* design document for details.

Configurations

The same TF-M code base is flexible to address different implementation requirements, from the simplest device with isolation level 1 to the most complicated device with isolation level 3 and optional isolation rules.

These configurations are set by switches, during the build time, as runtime support costs extra resources. The common configurations are named *profile*. There are several profiles defined.

History

Table 41: Revision

Date	Description
2021 Apr-Sep	Updated to cover the implementation for *FF-M v1.1* features.
2018	Created as 'TF-M Inter-Process Communication' which is deprecated as this document covers whole SPM content.

Copyright (c) 2021,2024, Arm Limited. All rights reserved.

Secure Partition Runtime Library

Organization
Arm Limited

Contact
tf-m@lists.trustedfirmware.org

Background

Trusted Firmware - M (TF-M) uses a toolchain provided runtime library and supervisor calls to easily implement the PSA Firmware Framework (PSA FF) API. This working model works well under isolation level 1 since there are no data isolation requirements. While TF-M is evolving, this model is not suitable because:

- The high-level isolation requires isolating data but some toolchain library interfaces have their own global data which cannot be shared between the Secure Partitions.
- The toolchain libraries are designed without taking security as a core design principle.

A TF-M specific runtime library is needed for the following reasons:

- Easier evaluation or certification by security standards.
- Source code transparency.
- Sharing code to save ROM and RAM space for TF-M.

PSA FF specification also describes the requirements of C runtime API for Secure Partitions.

This runtime library is named the Secure Partition Runtime Library, and the abbreviation is SPRTL.

Design Principle

The following requirements are mandatory for SPRTL implementation:

Important:

- **CODE ONLY** - No read-write data should be introduced into runtime library implementation.
- **Thread safe** - All functions are designed with thread-safe consideration. These APIs access caller stack and caller provided memory only.
- **Isolation** - Runtime API code is set as executable and read-only in higher isolation levels.
- **Security first** - SPRTL is designed for security and it may come with some performance loss.

API Categories

Several known types of functions are included in SPRTL:

- C runtime API.
- RoT Service API.
- PSA Client and Service API.
- [Future expansion, to be detailed later] other secure API.

Security Implementation Requirements

If malloc/realloc/free are provided, they must obey additional requirements compared to the C standard: newly allocated memory must be initialized to ZERO, and freed memory must be wiped immediately in case the block contains sensitive data.

The comparison API ('memcmp' e.g.), they should not return immediately when the fault case is detected. The implementation should execute in linear time based on input to avoid execution timing side channel attack.

The pointer validation needs to be considered. In general, at least the 'non-NULL' checking is mandatory. A detection for invalid pointer leads to a psa_panic().

The following section describes the first 3 API types and the implementation requirements.

C Runtime API

PSA FF describes a small set of the C standard library. Part of toolchain library API can be used as default if these APIs meet the *Design Principle* and *Security Implementation Requirements*. The toolchain 'header' and 'types' can be reused to simplify the implementation.

These APIs can take the toolchain provided version, or separately implemented in case there are extra requirements:

Note:

- 'memcpy()/memmove()/memset()'
- String API

These APIs are proposed to be implemented with the security consideration mentioned in *Security Implementation Requirements*:

Note:

- 'memcmp()'

- Other comparison API if referenced ('strcmp' e.g.).

The following functions are optional, but if present, they must conform to additional *Security Implementation Requirements*:

Note:

- 'malloc()/free()/realloc()'
- 'assert()/printf()'

The following APIs are coupled with toolchain library much so applying toolchain library implementation is recommended:

Note:

- Division and modulo - arithmetic operations.
- Other low level or compiler specific functions (such as 'va_list').

Besides the APIs mentioned above, the following runtime APIs are required for runtime APIs with private runtime context ('malloc' e.g.):

Note:

- '__sprtmain()' - partition entry runtime wrapper.

RoT Service API

The description of RoT Service API in PSA FF:

Note: Arm recommends that the RoT Service developer also defines an RoT Service API and implementation to encapsulate the use of the IPC protocol, and improve the usability of the service for client firmware.

Part of the RoT Service API have proposed specifications, such as the PSA Cryptography API, PSA Storage API, and PSA Attestation API. It is suggested that the service developer create documents of their RoT Service API and make them publicly available.

The RoT Service API has a large amount and it is the main part of SPRTL. This chapter describes the general implementation of the RoT Service API and the reason for putting them into SPRTL.

In general, a client uses the PSA Client API to access a secure service. For example:

```
/* Example, not a real implementation */
caller_status_t psa_example_service(void)
{
  ...
  handle = psa_connect(SERVICE_SID, SERVICE_VERSION);
  if (INVALID_HANDLE(handle)) {
    return INVALID_RETURN;
  }

  status = psa_call(handle, type, invecs, inlen, outvecs, outlen);

  psa_close(handle);
```

(continues on next page)

(continued from previous page)

```
  return TO_CALLER_STATUS(status);
}
```

This example encapsulates the PSA Client API, and can be provided as a simpler and more generic API for clients to call. It is not possible to statically link this API to each Secure Partition because of the limited storage space. The ideal solution is to put it inside SPRTL and share it to all Secure Partitions. This would simplify the caller logic into this:

```
if (psa_example_service() != STATUS_SUCCESS) {
  /* do something */
}
```

This is the simplest case of encapsulating PSA Client API. If a RoT Service API is connect heavy, then, the encapsulation can be changed to include a connection handle inside a context data structure. This context data structure type is defined in RoT Service headers and the instance is allocated by API caller since API implementation does not have private data.

Note:

- Even the RoT Service APIs are provided in SPRTL for all clients, the SPM performs the access check eventually and decides if the access to service can be processed.
- For those RoT Service APIs only get called by a specific client, they can be implemented inside the caller client, instead of putting it into SPRTL.

PSA Client and Service API

Most of the PSA APIs can be called directly with supervisor calls. The only special function is psa_call, because it has **6** parameters. This makes the supervisor call handler complex because it has to extract the parameters from the stack. The definition of psa_call is the following:

```
psa_status_t psa_call(psa_handle_t handle, int32_t type,
                      const psa_invec *in_vec, size_t in_len,
                      psa_outvec *out_vec, size_t out_len);
```

The parameters need to be packed to avoid passing parameters on the stack, and the supervisor call needs to unpack the parameters back to **6** for subsequent processing.

Privileged Access Supporting

Due to specified API (printf, e.g.) need to access privileged resources, TF-M Core needs to provide interface for the resources accessing. The permission checking must happen in Core while caller is calling these interface.

Secure Partition Local Storage

There are APIs that need to reference specific partition private data ('malloc' references local heap, e.g.), and the APIs reference the data by mechanisms other than function parameters. The mechanism in TF-M is called 'Secure Partition Local Storage'.

A straight way for accessing the local storage is to put the local storage pointer in a known position in the stack, but there is a bit of difficulty in particular scenarios.

Note:

- The partition's stack is not fixed-size aligned, using stack address aligning method can not work.

- It requires privileged permission to *access* special registers such as *PSPLIMIT*. And Armv6-M and Armv7-M don't have *PSPLIMIT*.'

Another common method is to put the pointer in one shared global variable, and the scheduler maintains the value of this variable to point to the running partition's local storage in runtime. It does not fully align with SPRTL design prerequisites listed above, hence extra settings are required to guarantee the isolation boundaries are not broken.

Important:

- This variable is put inside a dedicated shared region and it can not hold information not belonging to the owner.

And this mechanism has disadvantages:

- It needs extra maintenance effort from the scheduler and extra resources for containing the variable.

TF-M chooses this common way as the default option for local storage and can be expanded to support more methods.

Tooling Support on Partition Entry

PSA FF requires each Secure Partition to have an entry point. For example:

```
/* The entry point function must not return. */
void entry_point(void);
```

Each partition has its own dedicated local_storage for heap tracking and other runtime state. The local_storage is designed to be saved at the read-write data area of a partition with a specific name. A generic entry point needs to be available to get partition local_storage and do initialization before calling into the actual partition entry. This generic entry point is defined as '__sprtmain':

```
void ___sprtmain(void)
{
  /* Get current SP private data from local storage */
  struct p_sp_local_storage_t *m =
          (struct p_sp_local_storage_t *)tfm_sprt_local_storage;

  /* Potential heap init - check later chapter */
  if (m->heap_size) {
    m->heap_instance = tfm_sprt_heap_init(m->heap_sa, m->heap_sz);
  }

  /* Call thread entry 'entry_point' */
  m->thread_entry();

  /* Back to tell Core end this thread */
  SVC(THREAD_EXIT);
}
```

Since SPM is not aware of the '__sprtmain' in SPRTL, it just calls into the entry point listed in partition runtime data structure. And the partition writer may be not aware of running of '__sprtmain' as the generic wrapper entry, tooling support needs to happen to support this magic. Here is an example of partition manifest:

```
{
  "name": "TFM_SP_SERVICE",
  "type": "PSA-ROT",
```

(continues on next page)

(continued from previous page)

```
  "priority": "NORMAL",
  "entry_point": "tfm_service_entry",
  "stack_size": "0x1800",
  "heap_size": "0x1000",
  ...
}
```

Tooling would do manipulation to tell SPM the partition entry as '__sprtmain', and TF-M SPM would maintain the local storage at run time. Finally, the partition entry point gets called and run, tooling helps on the decoupling of SPM and SPRTL implementation. The pseudo code of a tooling result:

```
struct partition_t sp1 {
  .name = "TFM_SP_SERVICE",
  .type = PSA_ROT,
  .priority = NORMAL,
  .id = 0x00000100,
  .entry_point = ___sprtmain, /* Tell SPM entry is '___sprtmain' */
  .local_storage = { /* struct sprt_local_storage_t */
    .heap_sa = sp1_heap_buf,
    .heap_sz = sizeof(sp1_heap_buf),
    .thread_entry = sp1_entry, /* Actual Partition Entry */
    .heap_instance = NULL,
  },
}
```

Implementation

The SPRTL C Runtime sources are put under: '$TFM_ROOT/secure_fw/partitions/lib/runtime/'

The output of this folder is a static library named as 'libtfm_sprt.a'. The code of 'libtfm_sprt.a' is put into a dedicated section so that a hardware protected region can be applied to contain it.

The RoT Service API are put under service interface folder. These APIs are marked with the same section attribute where 'libtfm_sprt.a' is put.

The Formatting API - 'printf' and variants

The 'printf' and its variants need special parameters passing mechanism. To implement these APIs, the toolchain provided builtin macro 'va_list', 'va_start' and 'va_end' cannot be avoided. This is because of some scenarios such as when 'stack canaries' are enabled, only the compiler knows the format of the 'canary' in order to extract the parameters correctly.

To provide a simple implementation, the following requirements are defined for 'printf':

- Format keyword 'xXduscp' needs to be supported.
- Take '%' as escape flag, '%%' shows a '%' in the formatted string.
- To save heap usage, 32 bytes buffer in the stack for collecting formatted string.
- Flush string outputting due to: a) buffer full b) function ends.

The interface for flushing can be a logging device.

Function needs implied inputs

Take 'malloc' as an example. There is only one parameter for 'malloc' in the prototype. Heap management code is put in the SPRTL for sharing with caller partitions. The heap instance belongs to each partition, which means this instance needs to be passed into the heap management code as a parameter. For allocation API in heap management, it needs two parameters - 'size' and 'instance', while for 'malloc' caller it needs a 'malloc' with one parameter 'size' only. As

mentioned in the upper chapter, this instance can be retrieved from the Secure Partition Local Storage. The implementation can be:

```
void *malloc(size_t sz)
{
    struct p_sp_local_storage_t *m =
        (struct p_sp_local_storage_t *)tfm_sprt_local_storage;

    return tfm_sprt_alloc(m->heap_instance, sz);
}
```

Copyright (c) 2019-2024, Arm Limited. All rights reserved.

TF-M Inter-Process Communication

Authors
Ken Liu, Mingyang Sun

Organization
Arm Limited

Contact
ken.liu@arm.com, mingyang.sun@arm.com

Terminology

IPC - Inter-Process Communication

For more terminology please check *Reference* document.

Design Overview

Components for implementing IPC:

- SPM – for partition information and isolation actions
- Core – for exception handling
- Memory pool
- Message manager
- Thread
- Synchronization objects
- PSA API

Implementation Details

Listed modules are all internal modules except PSA API. Prototypes and definitions are not listed for internal modules in this document. For PSA API definitions, check them in PSA Firmware Framework specification in the reference chapter.

SPM and Core

SPM manages Secure Partition information. Enhancements need to be done in SPM data structure for Secure Partition for IPC due to:

- IPC model requires each Secure Partition has its own stack area.
- Multiple services are holding in same Secure Partition and each service has its own information like message queue, SID and priority.
- Changed information related manifest items need to be changed, too.

Modifications in Core:

- More SVC calls need to be added into list since PSA API are implemented as SVC calls in TF-M.
- New PendSV handler for thread scheduling.
- Arch-related context stacking and switching.

Memory Pool

Handles of connection and messages for Secure Partition needs to be allocated dynamically. A memory pool is provided in the system to handle dynamic allocation. Each memory pool item contains below information:

- A list iterator to chain all of memory pool items.
- An information member to record information like size and types.
- The memory item body for caller usage.

A memory area needs to be provided in SPM for the memory pool. It could be an array of memory areas defined in the linker script. Two chains are available to manage the items: free chain and used chain. And an LRU (Last recent used) mechanism is applied for fast seeking while item allocating and destroying.

Message Manager

Message Manager handles message creating, pushing, retrieving and destroy. A message contains below information:

- Message sender and destination
- Message status
- IO vectors for service
- 'psa_msg_t' for service

A checking needs to be performed in SPM before creating a message to detect if a message with the same sender and destination is ongoing. This avoids repeat messages are available in the queue.

Thread

Each Secure Partition has a thread as execution environment. Secure Partition is defined statically in TF-M manifest, which indicates that a number of threads are statically defined. Threads are chained in SPM and sorted with its priority, and there is an extra indicator point to first running thread with the highest priority. This helps fast seeking of running threads while the scheduler is switching threads.

Thread context contains below information:

- Priority
- Status
- Stack pointer
- Stack pointer limitation
- Entry
- Parameter
- Entry return value
- Context

- List iterator

Thread API provides below functions:

- Thread creating and destroying
- Thread status retrieving and changing
- Current thread retrieving
- Thread context switching

PendSV exception in TF-M core is the place thread context APIs been called. Before thread switching taking place, isolation status needs to be changed based on Secure Partition change and current isolation level – a thread is a member of partition which means thread switching caused a partition switching.

Synchronization API

A first synchronization object is an event. This could be applied into event waiting in the partition, and message response handling in IPC. The event object contains below members:

- Owner thread who is waiting for this event
- Event status (Ready or Not-Ready)
- List iterator for synchronization objects management

Event API Limitation: could be waited by one thread only.

PSA API

This chapter describes the PSA API in an implementation manner.

- API type: could be Client API and Service Partition API
- Block-able: Block-able API may block caller thread; Non-Block API does not block caller thread.
- Description: The functionality description and important comments.

```
uint32_t psa_framework_version(void);
uint32_t psa_version(uint32_t sid);
```

- Client API
- Non-Block API
- These 2 functions are finally handled in SPM and return the framework version or version to the caller.

```
psa_handle_t psa_connect(uint32_t sid, uint32_t version);
psa_status_t psa_call(psa_handle_t handle, int32_t type,
                      const psa_invec *in_vec, size_t in_len,
                      psa_outvec *out_vec, size_t out_len);
void psa_close(psa_handle_t handle);
```

- Client API
- Block-able API
- These 3 APIs are implemented in the same manner and just different parameters. SPM converts each call into a corresponding message with a parameter in the message body and pushes the message into service queue to wait for the response. Scheduler switches to a specified thread (partition) and makes Secure Partition to have chance retrieving and process message. After a message response is returned to the caller, the waiting caller gets to go and get the result.

```
psa_signal_t psa_wait(psa_signal_t signal_mask, uint32_t timeout);
```

- Secure Partition API
- Block-able API
- This API blocks caller partition if there is no expected event for it. This function is implemented based on event API.

```
void psa_set_rhandle(psa_handle_t msg_handle, void *rhandle);
psa_status_t psa_get(psa_signal_t signal, psa_msg_t *msg);
size_t psa_read(psa_handle_t msg_handle, uint32_t invec_idx,
                void *buffer, size_t num_bytes);
size_t psa_skip(psa_handle_t msg_handle, uint32_t invec_idx,
                size_t num_bytes);
void psa_write(psa_handle_t msg_handle, uint32_t outvec_idx,
               const void *buffer, size_t num_bytes);
void psa_reply(psa_handle_t msg_handle, psa_status_t status);
void psa_clear(void);
void psa_eoi(psa_signal_t irq_signal);
```

- Secure Partition API
- Non-Block
- These APIs do not take the initiative to change caller status. They process data and return the processed data back to the caller.

```
void psa_notify(int32_t partition_id);
```

- Secure Partition API
- Non-Block
- This API sets DOORBELL bit in destination partition's event. This API does not take the initiative to change caller status.

```
void psa_panic(void);
```

- Secure Partition API
- Block-able API
- This function will terminate execution within the calling Secure Partition and will not return.

Reference

PSA Firmware Framework specification URL

Copyright (c) 2019-2022, Arm Limited. All rights reserved.

Stateless Root of Trust Services Reference

Author
Mingyang Sun

Organization
Arm Limited

Contact
mingyang.sun@arm.com

Introduction

This document describes the implementation for the FF-M v1.1 feature - 'Stateless RoT Service', and the related references when developing RoT services.

It is recommended to refer to the FF-M v1.0 specification[1] and FF-M v1.1 extension[2] for background and rationale details.

Implementation Details

This chapter describes the implementation-defined items, including stateless handle value definition, tooling update, and programming API changes.

Stateless Handle Value Definition

The index, stateless indicator, and service version information are encoded into a handle by the manifest tooling, and then generated to header file sid.h.

Table 42: Bit Fields of Stateless Handle

Bits	Field Description
bit 31	reserved
bit 30	stateless handle indicator bit, always 1
bit 29 - bit 16	reserved
bit 15 - bit 8	service version requested by client - for client version check
bit 7 - bit 5	reserved
bit 4 - bit 0	the handle index, [0, 31]

Since connection-based services and stateless services share the same PSA API psa_call(), an indicator bit is set in the handle indicate the type of the handle. If it is set, the handle is stateless, and definition is as described in the table above. Maximum connection-based handle is 0x3FFFFFFF, thus the indicator bit is always 0.

The index is used for organizing stateless services in manifest tool and locating a stateless service in SPM logic. A range of index [0, 31] is the initial implementation. Future expansion is possible.

Tooling Support

TF-M provides a tool (tools/tfm_parse_manifest_list.py) to generate source header files required by partition and services. For example, the generated sid.h contains service ID and version. The tooling is extended to generate stateless handle from partition manifests automatically.

The stateless_handle attribute in manifest is only supported by partitions with firmware framework version 1.1.

- If stateless_handle in manifest is set to an integer, the index is stateless_handle - 1.
- If it is auto or not set, the first empty index in range [0, 31] is assigned.
- Other settings - tooling reports an error.

Finally, the tooling encodes the handle according to definitions in *Stateless Handle Value Definition* section, and writes them to sid.h header file.

Changes in Programming API

[1] FF-M v1.0 Specification
[2] FF-M v1.1 Extension

This chapter describes the changes in programming API for stateless services. The following APIs' behavious and message data structure members are updated to support the stateless service.

psa_connect()

According to FF-M v1.1, client calling psa_connect() with the SID of a stateless RoT Service is a PROGRAMMER_ERROR.

psa_close()

According to FF-M v1.1, client passing a stateless handle to call this API is a PROGRAMMER_ERROR.

psa_call()

```
psa_status_t psa_call(psa_handle_t handle, int32_t type,
                      const psa_invec *in_vec, size_t in_len,
                      psa_outvec *out_vec, size_t out_len)
```

API parameters and behaviour change:

1. The handle parameter must be a stateless handle defined in psa_manifest/sid.h when requesting a stateless service.
2. This API validates stateless handle, decodes index and service version information from it. SPM uses the index to know which stateless service is requested.
3. This API performs some operations as psa_connect() does, such as the authorization check, service and client version check, and handle space allocation.

Service behaviour change during a "psa_call":

Service does not accept connection and disconnection messages. After a "psa_call" request is serviced, it calls psa_reply(), frees the connection handle to handle pool.

psa_set_rhandle()

According to FF-M v1.1, stateless service calling this API on a message is a PROGRAMMER_ERROR and it will never return.

psa_msg_t type

The rhandle member of a psa_msg_t type received by a stateless RoT Service is always NULL.

Application Recommendation

There are particular services that do not need sessions. The access to the service is a one-shot connection. These services provide standalone operations that do not require any non-volatile state of resources to be maintained by the RoT service itself or do not expose any kind of context or session to the caller. Such services are recommended to be implemented as stateless, to provide quick access and to avoid extra overheads.

In this case, rhandle access would be prohibited as it is used for saving state or non-volatile resources while stateless services do not need them.

Update Feasibility of Existing Services

TF-M native services are used widely. They only need standalone operations and do not need to keep state between sessions. For example, the service in Crypto partition does not do anything during psa_connect() or psa_close() process. Same for services in other partitions, thus all of them can be implemented as stateless.

Analysis for them:

Table 43: TF-M Partition Services Update Possibility

Partition	Number of Services	Can be Stateless
ITS	4	All
PS	5	All
Crypto	1	All
FWU	6	All
Platform	4	All
Initial Attestation	2	All

Other services are not analyzed here.

Grouping Services

Stateless service table is stored statically, and TF-M supports 32 stateless services currently.

Similar stateless services in a partition could be grouped, and assign one SID for the group. The type parameter in psa_call() could be extended to identify the service in group. In this case, it is recommended to use consecutive values for type.

It is recommended that each Secure Partition declares one stateless service and uses the type parameter to distinguish different stateless services. Therefore, more stateless services can be supported.

Migrating to Stateless Services

Please refer to Chapter 4 “Stateless Root of Trust services”, Appendix B.3.2 “Using a stateless RoT Service”, and Appendix D “Implementing session-less RoT Services” in FF-M v1.1 document for details on which kind of service can be stateless and how to implement a stateless service.

Reference

Copyright (c) 2021-2024, Arm Limited. All rights reserved.

Uniform Secure Service Signature

Author
Miklos Balint

Organization
Arm Limited

Contact
Miklos Balint <miklos.balint@arm.com>

Declaring secure service interface

The following alternative secure service signature is proposed as an amendment to existing implementation.

Individual signatures - current method

A <service_name>_veneers.c file is created in the secure_fw/ns_callable directory, that specifies the signature for each veneer function, and calls the secure function from the veneers. The respective interface/include/<service_name>_veneers.h file with the veneer declarations have to be created and maintained manually. Note that at present TF-M framework limits the range of valid return values a secure service can provide, reserving a range for framework error codes.

Uniform signatures - proposal

The proposal is to use a uniform signature for all the secure functions of the secure service. There are multiple advantages of this method:

- TF-M Core can do a sanity check on the access rights of the veneer parameters, and there is no need for the secure services to make these checks individually. Please note that in the present implementation sanity check is only fully supported for level 1 isolation.
- The veneer declarations and implementations for the secure functions can be generated automatically from a template (using the secure function list in the secure service's manifest)

The signature for such secure services would look like this:

```
psa_status_t secure_function_name(struct psa_invec *in_vec, size_t in_len,
                                  struct psa_outvec *out_vec, size_t out_len);
```

where

Return value:

psa_status_t is a status code whose values are described in PSA Firmware Framework (as in version 1.0-beta-0 chapter 4.3.3).

Note:

The return value limitations imposed by TF-M framework for proprietary secure service veneers would not apply to secure services using the uniform signature. This is analogous to how PSA Firmware Framework handles values returned by psa_reply() function.

Arguments:

```
/**
 * A read-only input memory region provided to a RoT Service.
 */
typedef struct psa_invec {
    const void *base;   /*!< the start address of the memory buffer */
    size_t len;         /*!< the size in bytes */
} psa_invec;

/**
 * A writable output memory region provided to a RoT Service.
 */
typedef struct psa_outvec {
    void *base;         /*!< the start address of the memory buffer */
    size_t len;         /*!< the size in bytes */
} psa_outvec;

/**
 * in_len: the number of input parameters, i.e. psa_invecs
 * out_len: the number of output parameters, i.e. psa_outvecs
 */
```

The number of vectors that can be passed to a secure service is constrained:

```
in_len + out_len <= PSA_MAX_IOVEC
```

The veneer function declarations and implementations are generated in the interface/include/tfm_veneers.h and secure_fw\ns_callable\tfm_veneers.c files respectively. The veneer functions are created with the name tfm_<secure_function_name>_veneer

Services that implement the uniform signature do not need to manually fill the template veneer function to call TFM_CORE_SFN_REQUEST macro.

Compatibility

Note that the proposal is for the two types of services (those with proprietary signatures and those with uniform signatures) to co-exist, with the intention of eventually phasing out proprietary signatures in favour of the more robust, uniform signature.

Copyright (c) 2019-2020, Arm Limited. All rights reserved.

TF-M Crypto Service design

Author
Antonio de Angelis

Organization
Arm Limited

Contact
Antonio de Angelis <antonio.deangelis@arm.com>

Table of Contents

- *TF-M Crypto Service design*
  - *Abstract*
  - *Introduction*
  - *Components*
  - *Relationship between Mbed TLS and the TF-M Crypto service*
    - *TF-M Crypto as a particular configuration of Mbed TLS*
    - *Usage of Mbed TLS configuration headers*
    - *Hardware acceleration*
    - *Builtin keys*
  - *Service API description*
  - *Configuration parameters*
  - *References*

Abstract

This document describes the design of the TF-M Cryptographic Secure Service (in short, TF-M Crypto service).

Introduction

The TF-M Crypto service provides an implementation of the PSA Certified Crypto APIs in a PSA RoT secure partition in TF-M. It is based on the Mbed TLS project, which provides a reference implementation of the PSA Crypto APIs as a *C* software library. For more details on the PSA Crypto APIs refer to[1], while for the Mbed TLS reference software refer to[2] and[3]

[1] PSA Certified Crypto API specifications: https://arm-software.github.io/psa-api/crypto/
[2] Using PSA - Getting started in Mbed TLS: https://mbed-tls.readthedocs.io/en/latest/getting_started/psa/
[3] Mbed TLS repository which holds the reference implementation as a *C* software library: https://github.com/Mbed-TLS

The service can be requested by other services running in the SPE, or by applications running in the NSPE, and its aim is to provide cryptographic primitives in a secure and efficient way, either via software or by routing the calls to any underlying crypto hardware accelerator or secure element that the platform might provide.

Components

The TF-M Crypto service is implemented by a number of different firmware components residing in the Crypto partition, which are listed below:

Table 44: Components table

Component name	Description	Location
Client API interface	This module exports the PSA Crypto API to be callable from the users, which are called also *clients*. They could be either other secure partitions or Non Secure world based callers.	interface/src/tfm_crypto_api.c
Mbed TLS libmbedcrypto.a	The Mbed TLS libmbedcrypto.a library is used in the service as a cryptographic *backend* library which provides the APIs to implement a PSA Crypto *core*, SW based crypto primitives and wrappers for HW crypto accelerators and Secure Elements. It exposes those through the PSA Crypto APIs	Needed as dependency specified by the MBEDCRYPTO_PATH CMake configuration option
Init module	This module handles the initialisation of the service objects during TF-M boot and provides the infrastructure to service requests when TF-M is built for IPC or SFN model. The dispatching mechanism of IPC requests is based on a look up table of function pointers. This design allows for better scalability and support of a higher number of Secure functions with minimal overhead and duplication of code. This module is in charge of providing an ID of the caller of each API in the backend, allowing to enforce key ownership policies.	secure_fw/partitions/crypto/crypto_init.c
Alloc module	This module handles the allocation of contexts for multipart operations in the Secure world. This is required because the caller view of contexts, i.e. *clients*, does not contain any sensible information but just a number handle which is then used by the service itself to match the context to the actual context information which will be stored securely in the TF-M crypto partition private memory. This is enabled by setting MBEDTLS_PSA_CRYPTO_CLIENT option on the caller side. Note that setting this option on the client side is a hard requirement in order for the clients to work correctly	secure_fw/partitions/crypto/crypto_alloc.c
Service modules	These modules (AEAD, Asymmetric, Cipher, Hash, Key Derivation, Key Management, MAC, Random Number Generation) represent a thin layer which is in charge of servicing the calls from the clients. They provide parameter sanitization and context retrieval for multipart operations, and dispatching to the corresponding backend library function exposed by the underlying library.	secure_fw/partitions/crypto/crypto_aead.c secure_fw/partitions/crypto/crypto_asymmetric.c secure_fw/partitions/crypto/crypto_cipher.c secure_fw/partitions/crypto/crypto_hash.c secure_fw/partitions/crypto/crypto_key_derivation.c secure_fw/partitions/crypto/crypto_key_management.c secure_fw/partitions/crypto/crypto_mac.c secure_fw/partitions/crypto/crypto_rng.c
Backend library abstraction	This module contains several APIs to abstract the interface towards the backend library, which must provide the PSA Crypto core layer, key management, SW based crypto and possibly interfaces for HW crypto accelerators and Secure Elements	secure_fw/partitions/crypto/crypto_library.*
Manifest	The manifest file is a description of the service components.	secure_fw/partitions/crypto/tfm_crypto.yaml

The interaction between the different components is described by the block diagram in block_diagram-label

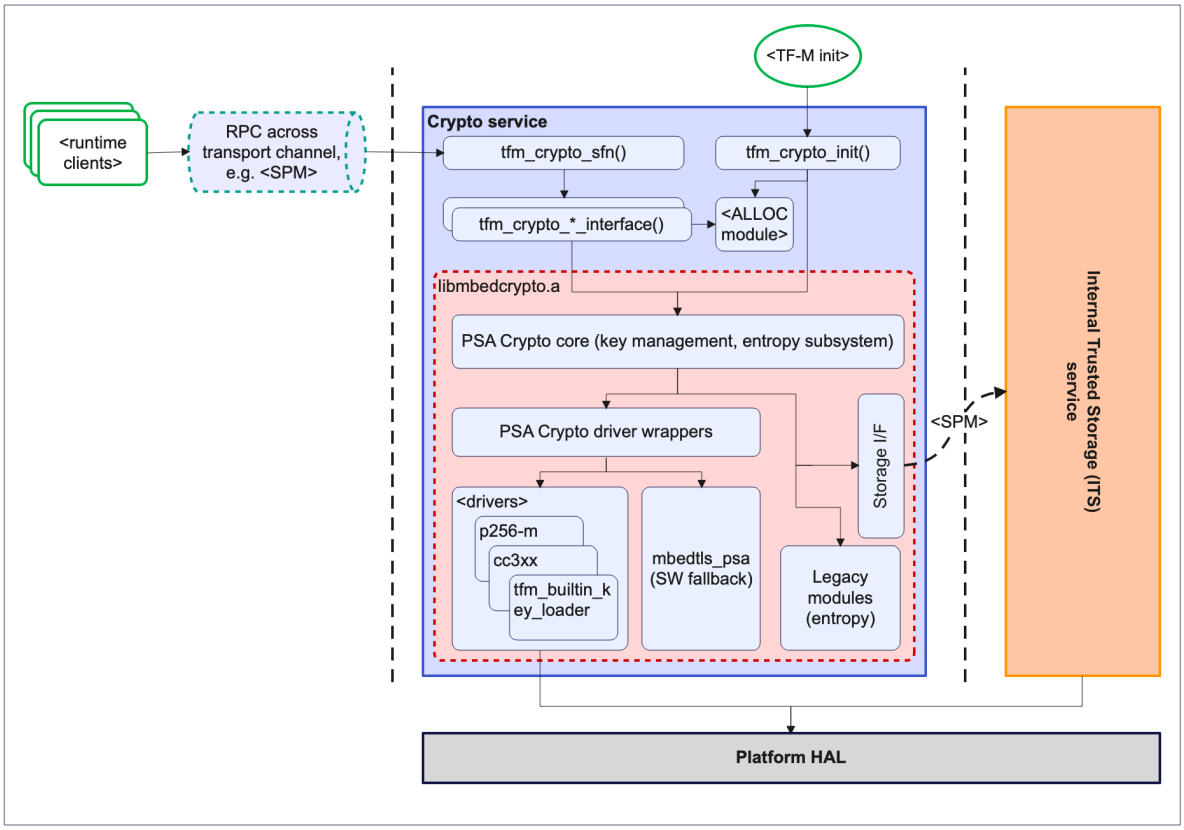


Fig. 12: Block diagram of the firmware architecture of the TF-M Crypto service. Dotted lines between services represent isolation boundaries once runtime firmware is initialized, i.e. TF-M init phase has completed. The diagram is simplified and shows only the major functional blocks, for a more detailed libmbedcrypto.a architecture please refer toPage 1411, 3.

Relationship between Mbed TLS and the TF-M Crypto service

TF-M Crypto as a particular configuration of Mbed TLS

Up until *TF-Mv2.0*, the TF-M Crypto service used to provide its own separate implementation of the PSA Certified Crypto APIs, i.e. it provided its own version of the implementation defined aspects of the specifications. Starting from *TF-Mv2.1*, the TF-M Crypto service fully aligns to the implementation defined by the Mbed TLS project, i.e. its implementation defined aspects are the same as the ones defined by Mbed TLS.

As a consequence, starting from *TF-Mv2.1* the PSA Crypto headers available in TF-M are a copy of those distributed by the Mbed TLS project. TF-M just uses them and won't accept any contribution to them, as those need to be discussed in the scope of the Mbed TLS project.

TF-M then represents just a configuration of the Mbed TLS reference implementation where the TF-M Crypto APIs are provided as a remote call across a transport channel, which might be represented by a TrustZone boundary (in Armv8.x-M systems), by a mailbox channel in heterogeneous systems, e.g. Cortex-A + Cortex-M systems, by an SPM mediated interface, e.g. partition to partition calls or, in general, through a mechanism which provides process separation between the client and the service sides of the API. In this context, the client must always define the Mbed TLS config option MBEDTLS_PSA_CRYPTO_CLIENT, while the service must always have MBEDTLS_PSA_CRYPTO_SPM, mainly to avoid symbol clashing at link time between the library interface and the tfm_crypto_api.c interface. When there is a component on

the service side which is able to identify the client through an ID, it is recommended to also define MBEDTLS_PSA_CRYPTO_KEY_ID_ENCODES_OWNER option in order to provide separation in the key space.

Usage of Mbed TLS configuration headers

Mbed TLS uses two different configuration headers, specified through the setting of the MBEDTLS_CONFIG_FILE, i.e. Mbed TLS config, and the setting of MBEDTLS_PSA_CRYPTO_CONFIG_FILE, i.e. the PSA configuration. In order to be able to perform header inclusion for psa/crypto.h, the configuration files must be visible to the compilation unit through the include hierarchy. If none of the macros are defined, the fall back strategy is to include the default config files available in the Mbed TLS repo, i.e. include/mbedtls/mbedtls_config.h and include/psa/crypto_config.h, which contain a set of default values for the macros.

Usage of the default header config when using the TF-M Crypto service is highly discouraged, mainly because both on the client side and on the service side a set of options must always be defined (or undefined), as described in the previous section. TF-M provides example _profiles_ which show the options and how they should be used on both client and service side of the integration. Note that to avoid falling back to the default PSA configuration, the Mbed TLS config file must always define the symbol MBEDTLS_PSA_CRYPTO_CONFIG. The symbol to enable the Mbed TLS config MBEDTLS_CONFIG_FILE instead must be available to the unit being compiled which is including psa/crypto.h, i.e. passed by the build system config stage.

Hardware acceleration

The TF-M Crypto partition must handle all HW related crypto tasks, if the platform is capable of offering hardware acceleration or if a complete Secure Element is present. The main difference between the two is that a hardware accelerator does not store keys but just accelerates operations, while a Secure Element is capable of storing keys and the PSA Crypto core running on the host must interface with it to store, retrieve or use them for crypto tasks, etc.

There are currently two methods to interface an accelerator into the Crypto service, and both rely on the Crypto partition fully owning the Crypto hardware, i.e. the memory mapped IO space must be bound the Crypto partition only. Both methods are implemented through the capability of the _backend_ library to either:

1. Provide a link time mechanism to replace pure SW implementations for algorithms with HW assisted implementations. In this case, the TF-M platform provides some additional HW abstraction through the usage of crypto_hw_accelerator_*() APIs. This is dubbed the *_ALT* approach and will be soon to be deprecated potentially starting from the release of Mbed TLS 4.0
2. Provide a cleanly defined interface specification[4] to describe the APIs that a driver must expose to the PSA Crypto core in order for the core to be able to offload operations to hardware. This is the preferred method for interfacing with HW.

Both solutions are currently handled at build time (either compilation or linking) by Mbed TLS. For details on how to integrate a driver please refer directly to the documentation referenced above and to the Mbed TLS repo.

Builtin keys

A particular driver using the interface described inPage 1415, 4 is the TF-M Builtin Key Loader driver[5]. The goal of the driver is to make Mbed TLS aware of *transparent builtin keys*, i.e. keys which can be read from the core (i.e. not fully opaque keys), but that are normally bound to the platform and provisioned in it, for which it would be more appropriate to treat them as standard

[4] PSA Unified Driver Interface for Cryptoprocessors: https://github.com/Mbed-TLS/mbedtls/blob/development/docs/proposed/psa-driver-interface.md

[5] TF-M Builtin Key Loader driver, normally described as *tfm_builtin_key_loader*

transparent keys. The concept of *transparent builtin keys* is not defined in the spec so it is specifically a non standard extension added by TF-M to the Mbed TLS implementation, which might be changed between releases until a standard solution is adopted. TF-M patches Mbed TLS on the fly to enable such behaviour using patches available in lib/ext/mbedcrypto. Implementations might disable the tfm_builtin_key_loader and then must provide their own alternative storage location for all of the TF-M required builtin keys, e.g. by having them stored in a Secure Element with a corresponding opaque driver.

Service API description

The Alloc and Init modules implement public APIs which are specific to the TF-M Crypto service, and are available only internally to other components of the TF-M Crypto partition. For a detailed description of the prototypes please refer to the tfm_crypto_api.h header.

Table 45: Init and Alloc modules APIs

Function	Module	Calle	Scope
tfm_cry	Init	SPM	Called during TF-M boot for initialisation. It does modules initialisation (it initializes the Alloc module) and initializes the *backend* library. Being the partition enabled for the SFN model, it does not implement any IPC specific message handler, instead it relies on the SPM being able to schedule SFN partitions using the SFN dispatcher with little overhead
tfm_cry	Init	SPM	Function to handle an SFN request or to interface with the message handler when running in IPC model
tfm_cry	Alloc	Init	Called by tfm_crypto_init(), it initialises the internal memory storage in the TF-M Crypto partition that the service uses to store multipart operation contexts as requested by clients.
tfm_cry	Alloc	Service modules	It allocates a new operation context for a multipart operation. It returns an handle to the allocated context in secure memory.
tfm_cry	Alloc	Service modules	It retrieves a previously allocated operation context of a multipart operation, based on the handle given as input.
tfm_cry	Alloc	Service modules	It releases a previously allocated operation context of a multipart operation, based on the handle given as input.
tfm_cry	*	Init	Interface functions called by the dispatcher to service PSA Crypto APIs requests

Configuration parameters

The TF-M Crypto service exposes some configuration parameters to tailor the service configuration in terms of supported functionalities and hence FLASH/RAM size to meet the requirements of different platforms and use cases. These parameters can be provided via CMake parameters during the CMake configuration step and as a configuration header to allow the configuration of the Mbed TLS library. When using Kconfig they are also exported in the Kconfig menus.

Table 46: Configuration parameters table

Parameter	Type	Description	Default
*CRYPTO_*	CMake build configuration parameter	Buffer used by Mbed TLS for its own allocations at runtime. This is a buffer allocated in static memory.	8096 (bytes)
*CRYPTO_*	CMake build configuration parameter	This parameter defines the maximum number of possible concurrent operation contexts (cipher, MAC, hash and key deriv) for multi-part operations, that can be allocated simultaneously at any time.	8
*CRYPTO_*	CMake build configuration parameter	This parameter applies only to IPC model builds. In IPC model, during a Service call, input and outputs are allocated temporarily in an internal scratch buffer whose size is determined by this parameter.	5120 (bytes)
*CRYPTO_*	CMake build configuration parameter	Defines the stack size assigned to the crypto partition in higher level of isolation configurations (L1 isolation has a common stack shared by all partitions)	6912 (bytes)
*CRYPTO_*	CMake build configuration parameter	Uses the Mbed TLS Crypto NV seed feature to provide entropy in case there is no HW acceleration providing HW entropy	Defined for platforms which don't have CRYPTO_HW_ACCELERATOR
*CRYPTO_*	CMake build configuration parameter	Defines the size of scratch buffers to handle input/outputs if the Memory Mapped IOVEC feature is not enabled	5120 (bytes)
*CRYPTO_*	CMake build configuration parameter	When enabled, only the multipart, i.e. non-integrated APIs will be available in the service	Not defined (Profile default)
*CRYPTO_*	CMake build configuration parameters	When enabled, the correspoding shim layer module and relative APIs are available in the service	Defined (Profile default)
MBEDTLS	Configuration header	The Mbed TLS library can be configured to support different algorithms through the usage of a configuration header file at build time. This allows for tailoring FLASH/RAM requirements for different platforms and use cases.	lib/ext/mbedcrypto/mbedcrypto_config/tfm_mbedcrypto_config_default.h (Profile default)
MBEDTLS	Configuration header	This header file specifies which cryptographic mechanisms are available through the PSA API when *MBEDTLS_PSA_CRYPTO_CONFIG* is enabled, and is not used when *MBEDTLS_PSA_CRYPTO_CONFIG* is disabled. Configuring TF-M always involves having the define enabled.	lib/ext/mbedcrypto/mbedcrypto_config/crypto_config_default.h (Profile default)

References

Copyright (c) 2019-2024, Arm Limited. All rights reserved.

Symmetric key algorithm based Initial Attestation

Author
David Hu

Organization
Arm Limited

Contact
david.hu@arm.com

Introduction

This document proposes a design of symmetric key algorithm based Initial Attestation in TF-M.

Symmetric key algorithm based Initial Attestation (*symmetric Initial Attestation* for short) signs and verifies Initial Attestation Token (IAT) with a symmetric cryptography signature scheme, such as HMAC. It can reduce TF-M binary size and memory footprint on ultra-constrained devices without integrating asymmetric ciphers.

This proposal follows PSA Attestation API document[1].

Note: As pointed out by PSA Attestation API[Page 1418, 1], the use cases of Initial Attestation based on symmetric key algorithms can be limited due to the associated infrastructure costs for key management and operational complexities. It may also restrict the ability to interoperate with scenarios that involve third parties.

Design overview

The symmetric Initial Attestation follows the existing IAT generation sequence for Initial Attestation based on asymmetric key algorithm (*asymmetric Initial Attestation* for short).

As Profile Small design[2] requests, a configuration flag SYMMETRIC_INITIAL_ATTESTATION selects symmetric initial attestation during build.

The top-level design is shown in *Overall design diagram* below.

Symmetric Initial Attestation adds its own implementations of some steps in IAT generation in Initial Attestation secure service. More details are covered in *IAT generation in Initial Attestation secure service*.

The interfaces and procedures of Initial Attestation secure service are not affected. Refer to Initial Attestation Service Integration Guide[3] for details of the implementation of Initial Attestation secure service.

Symmetric Initial Attestation invokes t_cose library to build up COSE_Mac0 structure. COSE_Mac0 support is added to t_cose library in TF-M since official t_cose hasn't supported COSE_Mac0 yet. The design of COSE_Mac0 support is covered in *COSE_Mac0 support in t_cose*.

Note: The COSE_Mac0 implementation in this proposal is a prototype only for Proof of Concept so far. It may be replaced after t_cose officially supports COSE_Mac0 message.

---

[1] PSA Attestation API 1.0 (ARM IHI 0085)
[2] *Trusted Firmware-M Profile Small Design*
[3] *Initial Attestation Service Integration Guide*

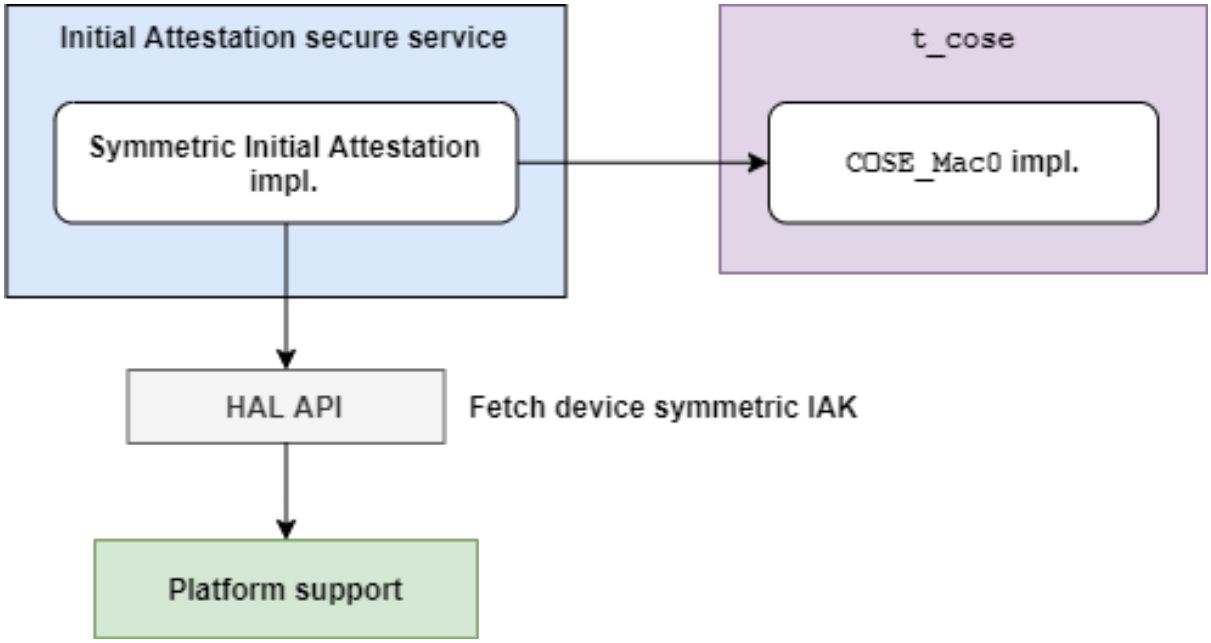


Fig. 13: Overall design diagram

Several HAL APIs are defined to fetch platform specific assets required by Symmetric Initial Attestation. For example, tfm_plat_get_symmetric_iak() fetches symmetric Initial Attestation Key (IAK). Those HAL APIs are summarized in *HAL APIs*.

Decoding and verification of symmetric Initial Attestation is also included in this proposal for regression test. The test suites and IAT decoding are discussed in *TF-M Test suite*.

QCBOR library and Crypto service are also invoked. But this proposal doesn't require any modification to either QCBOR or Crypto service. Therefore, descriptions of QCBOR and Crypto service are skipped in this document.

IAT generation in Initial Attestation secure service

The sequence of IAT generation of symmetric Initial Attestation is shown in *Symmetric IAT generation flow in Initial Attestation secure service* below. Note that the Register symmetric IAK stage is no longer required due to changes in the Crypto partition (attest_symmetric_key.c is now responsible only for calculating the instance ID).

In Initial Attestation secure service, symmetric Initial Attestation implements the following steps in attest_create_token(), which are different from those of asymmetric Initial Attestation.

- attest_token_start()
- Instance ID claims
- attest_token_finish()

If SYMMETRIC_INITIAL_ATTESTATION is selected, symmetric Initial Attestation dedicated implementations of those steps are included in build. Otherwise, asymmetric Initial Attestation dedicated implementations are included instead.

Symmetric Initial Attestation implementation resides a new file attest_symmetric_key.c to handle symmetric Instance ID related operations. Symmetric Initial Attestation dedicated attest_token_start() and attest_token_finish() are added in attestation_token.c.

The details are covered in following sections.

Symmetric Instance ID

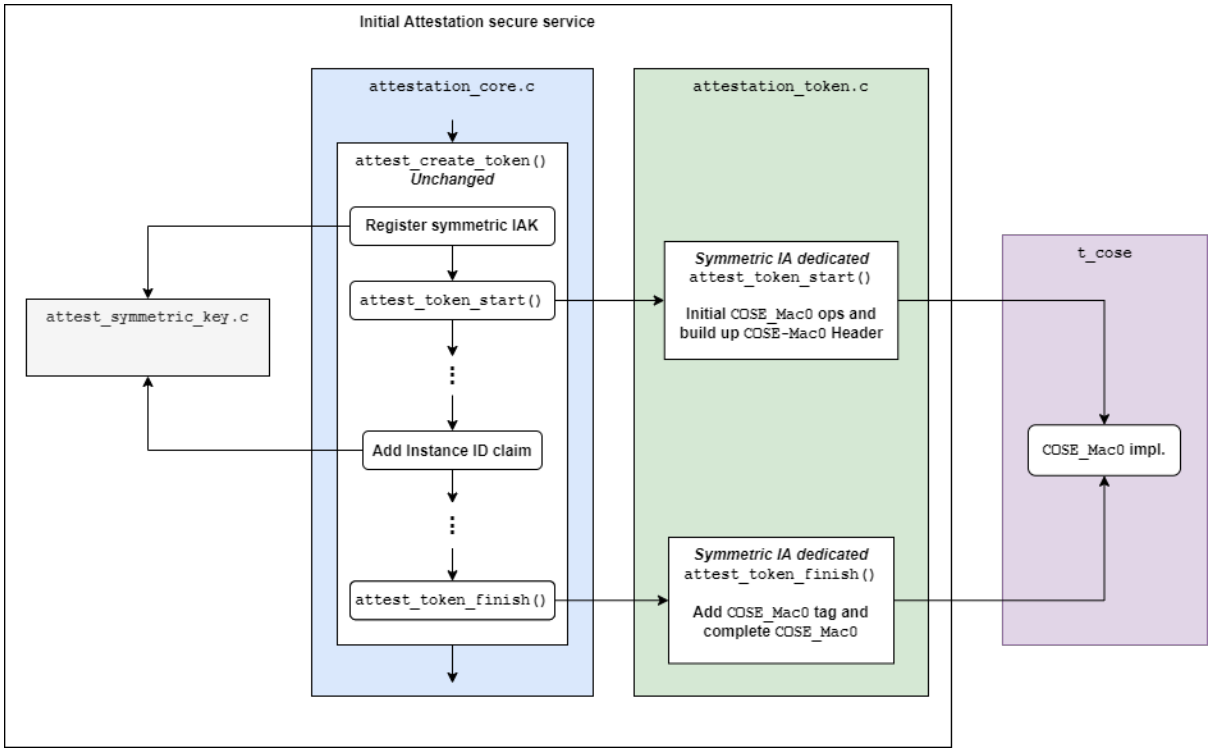


Fig. 14: Symmetric IAT generation flow in Initial Attestation secure service

Symmetric Initial Attestation dedicated attest_symmetric_key.c implements the attest_get_instance_id() function. This function returns the Instance ID value, calculating it if it has not already been calculated. Refer to *Instance ID claim_* for more details.

Note: Only symmetric IAK for HMAC algorithm is allowed so far.

Instance ID calculation

In symmetric Initial Attestation, Instance ID is also calculated the first time it is requested. It can protect critical symmetric IAK from being frequently fetched, which increases the risk of asset disclosure.

The Instance ID value is the output of hashing symmetric IAK raw data *twice*, as requested in PSA Attestation API[1]. HMAC-SHA256 may be hard-coded as the hash algorithm of Instance ID calculation.

Note: According to RFC2104[4], if a HMAC key is longer than the HMAC block size, the key will be first hashed. The hash output is used as the key in HMAC computation.

In current design, HMAC is used to calculate the authentication tag of COSE_Mac0. Assume that symmetric IAK is longer than HMAC block size (HMAC-SHA256 by default), the Instance ID is actually the HMAC key for COSE_Mac0 authentication tag generation, if Instance ID value is the output of hashing IAK only *once*. Therefore, attackers may request an valid IAT from device and fake malicious ones by using Instance ID to calculate valid authentication tags, to cheat others.

As a result, symmetric IAK raw data should be hashed *twice* to generate the Instance ID value.

The Instance ID calculation result is stored in a static buffer. Token generation process can call attest_get_instance_id() to fetch the data from that static buffer.

[4] HMAC: Keyed-Hashing for Message Authentication

attest_token_start()

Symmetric Initial Attestation dedicated attest_token_start() initializes the COSE_Mac0 signing context and builds up the COSE_Mac0 Header.

The workflow inside attest_token_start() is shown in *Workflow in symmetric Initial Attestation attest_token_start()* below.

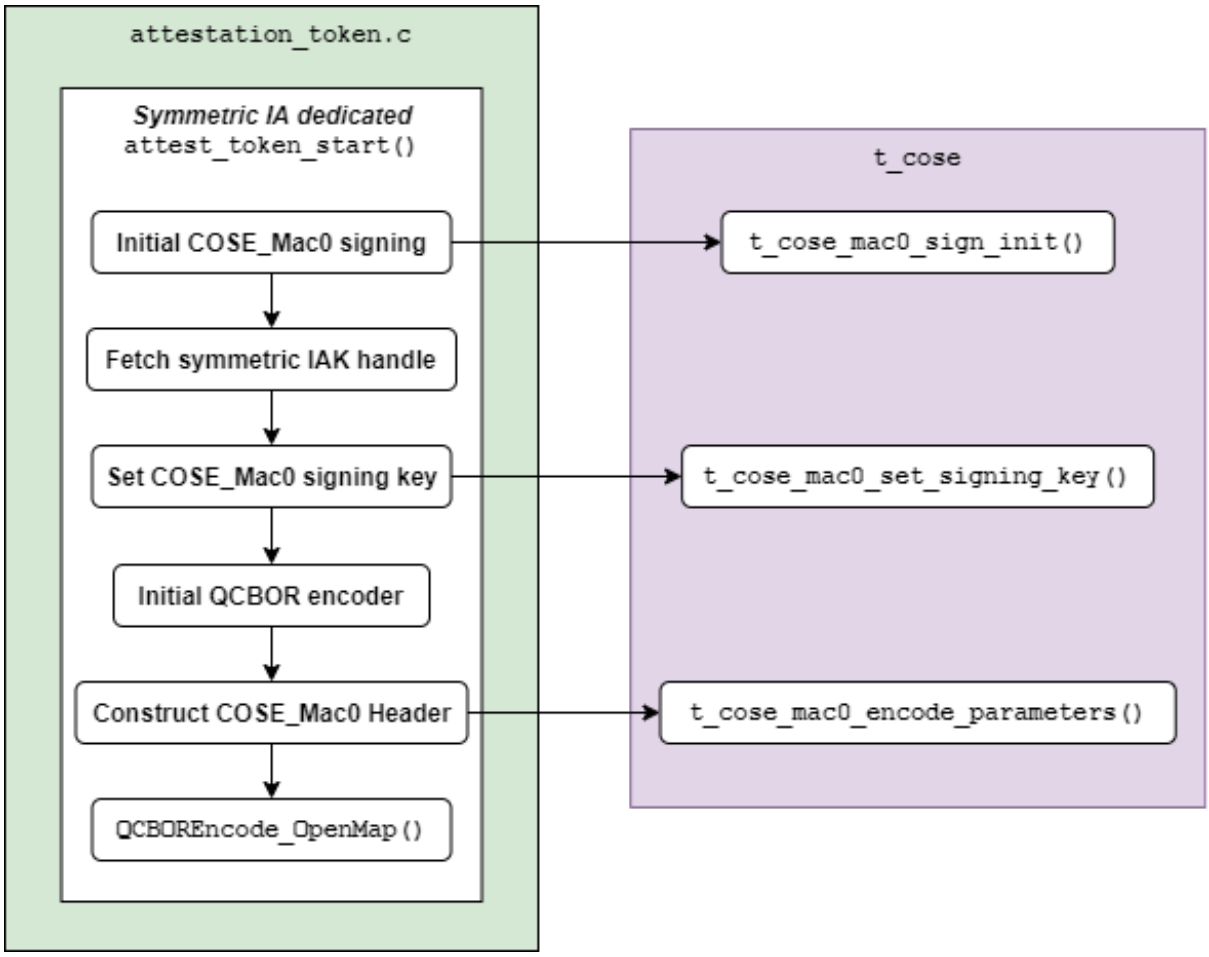


Fig. 15: Workflow in symmetric Initial Attestation attest_token_start()

Descriptions of each step are listed below:

1. t_cose_mac0_sign_init() is invoked to initialize COSE_Mac0 signing context in t_cose.
2. The symmetric IAK handle is set into COSE_Mac0 signing context via t_cose_mac0_set_signing_key().
3. Initialize QCBOR encoder.
4. The header parameters are encoded into COSE_Mac0 structure in t_cose_mac0_encode_parameters().
5. QCBOREncode_OpenMap() prepares for encoding the COSE_Mac0 payload, which is filled with IAT claims.

All the COSE_Mac0 functionalities in t_cose are covered in *COSE_Mac0 support in t_cose*.

Instance ID claim

Symmetric Initial Attestation also implements Instance ID claims in attest_add_instance_id_claim().

The Instance ID value is fetched via attest_get_instance_id(). The value has already been calculated during symmetric IAK registration. See *Instance ID calculation* for details.

The other steps are the same as those in asymmetric Initial Attestation implementation. The UEID type byte is set to 0x01.

attest_token_finish()

Symmetric Initial Attestation dedicated attest_token_finish() calls t_cose_mac0_encode_tag() to calculate and encode the authentication tag of COSE_Mac0 structure.

The whole COSE and CBOR encoding are completed in attest_token_finish().

The simplified flow in attest_token_finish() is shown in *Workflow in symmetric Initial Attestation attest_token_finish()* below.

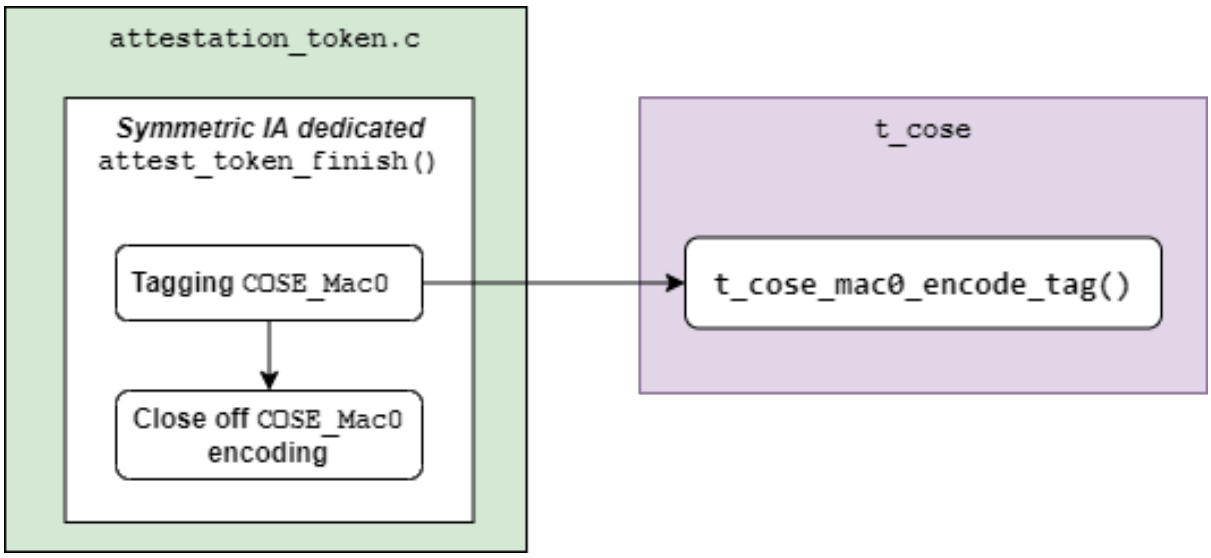


Fig. 16: Workflow in symmetric Initial Attestation attest_token_finish()

COSE_Mac0 support in t_cose

COSE_Mac0 supports in t_cose in TF-M include the following major functionalities:

- Encoding COSE_Mac0 structure
- Decoding and verifying COSE_Mac0 structure
- HMAC computation to generate and verify authentication tag
- Short-circuit tagging for test mode

According to RFC8152[5], COSE_Mac0 and COSE_Sign1 have similar structures. Therefore, the prototype follows COSE_Sign1 implementation to build up COSE_Mac0 file structure and implement COSE_Mac0 encoding and decoding.

Although COSE_Mac0 can share lots of data types, APIs and encoding/decoding steps with COSE_Sign1 in implementation, this prototype separates COSE_Mac0 implementation from COSE_Sign1. COSE_Mac0 owns its dedicated signing/verification contexts, APIs and encoding/decoding process. The purposes of separating COSE_Mac0 and COSE_Sign1 are listed below

- It can keep changes to COSE_Sign1 as small as possible and avoid conflicts with development in COSE_Sign1`. It can decrease conflicts if t_cose in TF-M is synchronized with original t_cose repository later.

[5] CBOR Object Signing and Encryption (COSE)

- COSE_Mac0 and COSE_Sign1 are exclusive in TF-M use cases. It cannot decrease TF-M memory footprint by extracting the common components shared by COSE_Mac0 and COSE_Sign1 but can make the design over-complicated.

Note: Only HMAC is supported in current COSE_Mac0 prototype.

File structure

New files are added to implement the functionalities listed above. The structure of files is shown in the table below.

Table 47: New files in t_cose

Directory	Files	Descriptions
src	t_cose_mac0_sig c	Encode COSE_Mac0 structure
	t_cose_mac0_ve c	Decode and verify COSE_Mac0 structure.
inc	t_cose_mac0_sig h	Data type definitions and function declarations of encoding and signing COSE_Mac0 message.
	t_cose_mac0_ve h	Data type definitions and function declarations of verifying COSE_Mac0 message.

Other t_cose files may also be changed to add COSE_Mac0 associated data types and function declarations.

HMAC operations are added in crypto_adapters/t_cose_psa_crypto.c. Preprocessor flags are added to select corresponding crypto for COSE message signing and verification.

- T_COSE_ENABLE_SIGN1 selects ECDSA and Hash operations for COSE_Sign1.
- T_COSE_ENABLE_MAC0 selects HMAC operations for COSE_Mac0.

Encoding COSE_Mac0

Following COSE_Sign1 implementation, COSE_Mac0 encoding exports similar functions to Initial Attestation secure service. The major functions are listed below.

Initialize signing context

t_cose_mac0_sign_init() initializes COSE_Mac0 signing context and configures option flags and algorithm used in signing.

```
static void
t_cose_mac0_sign_init(struct t_cose_mac0_sign_ctx *me,
                      int32_t                     option_flags,
                      int32_t                     cose_algorithm_id);
```

The COSE_Mac0 signing context is defined as

```
struct t_cose_mac0_sign_ctx {
    /* Private data structure */
    uint8_t              protected_parameters_buffer[
                           T_COSE_MAC0_MAX_SIZE_PROTECTED_PARAMETERS];
    struct q_useful_buf_c protected_parameters; /* The encoded protected parameters */
    int32_t              cose_algorithm_id;
    struct t_cose_key    signing_key;
```

(continues on next page)

(continued from previous page)

```
    int32_t                 option_flags;
    struct q_useful_buf_c kid;
    ...
};
```

Set signing key

t_cose_mac0_set_signing_key() sets the key used in COSE_Mac0 signing. Optional kid, as a key identifer, will be encoded into COSE_Mac0 Header unprotected bucket.

```
static void
t_cose_mac0_set_signing_key(struct t_cose_mac0_sign_ctx *me,
                            struct t_cose_key            signing_key,
                            struct q_useful_buf_c        kid);
```

Encode Header parameters

t_cose_mac0_encode_parameters() encodes the COSE_Mac0 Header parameters and outputs the encoded context to cbor_encode_ctx.

```
enum t_cose_err_t
t_cose_mac0_encode_parameters(struct t_cose_mac0_sign_ctx *context,
                              QCBOREncodeContext          *cbor_encode_ctx);
```

Calculate and add authentication tag

t_cose_mac0_encode_tag() calculates the authentication tag and finishes the COSE_Mac0 message.

```
enum t_cose_err_t
t_cose_mac0_encode_tag(struct t_cose_mac0_sign_ctx *context,
                       QCBOREncodeContext          *cbor_encode_ctx);
```

Decoding COSE_Mac0

Following COSE_Sign1 implementation, COSE_Mac0 decoding exports similar functions to test suite of Initial Attestation. The major functions are listed below.

Initialize verification context

t_cose_mac0_verify_init() initializes COSE_Mac0 verification context and configures option flags in verification.

```
static void
t_cose_mac0_verify_init(struct t_cose_mac0_verify_ctx *context,
                        int32_t                        option_flags);
```

The COSE_Mac0 verification context is defined as

```
struct t_cose_mac0_verify_ctx {
    /* Private data structure */
    struct t_cose_key      verification_key;
    int32_t                option_flags;
};
```

Set verification key

t_cose_mac0_set_verify_key() sets the key for verifying COSE_Mac0 authentication tag.

```
static void
t_cose_mac0_set_verify_key(struct t_cose_mac0_verify_ctx *context,
                           struct t_cose_key              verify_key);
```

Decode and verify COSE_Mac0

t_cose_mac0_verify() decodes the COSE_Mac0 structure and verifies the authentication tag.

```
enum t_cose_err_t
t_cose_mac0_verify(struct t_cose_mac0_verify_ctx *context,
                   struct q_useful_buf_c          cose_mac0,
                   struct q_useful_buf_c         *payload,
                   struct t_cose_parameters      *parameters);
```

Short-circuit tagging

If T_COSE_OPT_SHORT_CIRCUIT_TAG option is enabled, COSE_Mac0 encoding will hash the COSE_Mac0 content and add the hash output as an authentication tag. It is useful when critical symmetric IAK is unavailable or cannot be accessed, perhaps because it has not been provisioned or configured for the particular device. It is only for test and must not be used in actual use case. The kid parameter will either be skipped in COSE_Mac0 Header.

If T_COSE_OPT_ALLOW_SHORT_CIRCUIT option is enabled, COSE_Mac0 decoding will only verify the hash output, without requiring symmetric key for authentication tag verification.

TF-M Test suite

Symmetric Initial Attestation adds dedicated non-secure and secure test suites. The test suites also follow asymmetric Initial Attestation test suites implementation but optimize the memory footprint. Symmetric Initial Attestation non-secure and secure test suites request Initial Attestation secure service to generate IATs. After IATs are generated successfully, test suites decode IATs and parse the claims. Secure test suite also verifies the authentication tag in COSE_Mac0 structure.

Symmetric Initial Attestation implements its dedicated attest_token_decode_validate_token() in attest_symmetric_iat_decoded.c to perform IAT decoding required by test suites. If SYMMETRIC_INITIAL_ATTESTATION is selected, attest_symmetric_iat_decoded.c is included in build. Otherwise, asymmetric Initial Attestation dedicated implementations are included instead.

The workflow of symmetric Initial Attestation dedicated attest_token_decode_validate_token() is shown below.

If the decoding is required from secure test suite, attest_token_decode_validate_token() will fetch symmetric IAK to verify the authentication tag in COSE_Mac0 structure. If the decoding is required from non-secure test suite, attest_token_decode_validate_token() will decode COSE_Mac0 only by setting T_COSE_OPT_DECODE_ONLY option flag. Non-secure must not access the symmetric IAK.

HAL APIs

HAL APIs are summarized below.

Fetch device symmetric IAK

tfm_plat_get_symmetric_iak() fetches device symmetric IAK.

```
enum tfm_plat_err_t tfm_plat_get_symmetric_iak(uint8_t *key_buf,
                                               size_t buf_len,
                                               size_t *key_len,
                                               psa_algorithm_t *key_alg);
```

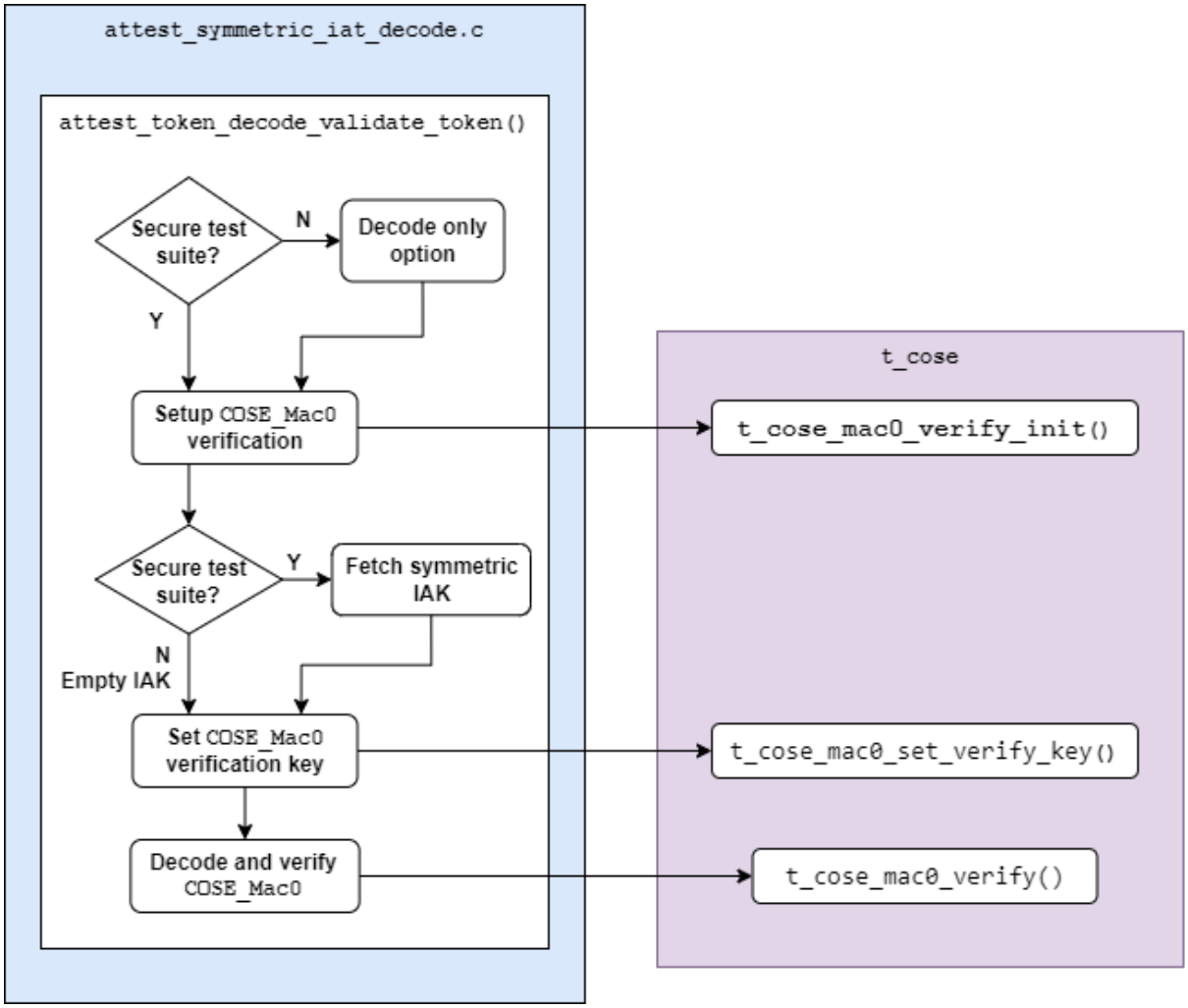


Fig. 17: Workflow in symmetric Initial Attestation attest_token_decode_validate_token()

Parameters:

key_buf	Buffer to store the symmetric IAK.
buf_len	The length of key_buf.
key_len	The length of the symmetric IAK.
key_alg	The key algorithm. Only HMAC SHA-256 is supported so far.

It returns error code specified in enum tfm_plat_err_t.

Get symmetric IAK key identifier

attest_plat_get_symmetric_iak_id() gets the key identifier of the symmetric IAK as the kid parameter in COSE Header.

Optional if device doesn't install a key identifier for symmetric IAK.

```
enum tfm_plat_err_t attest_plat_get_symmetric_iak_id(void *kid_buf,
                                                     size_t buf_len,
                                                     size_t *kid_len);
```

Parameters:

kid_buf	Buffer to store the IAK identifier.
buf_len	The length of kid_buf.
kid_len	The length of the IAK identifier.

It returns error code specified in enum tfm_plat_err_t.

Reference

Copyright (c) 2020-2022 Arm Limited. All Rights Reserved.

Internal Trusted Storage (ITS) Service

Author
Jamie Fox

Organization
Arm Limited

Contact
Jamie Fox <jamie.fox@arm.com>

Add support for block-aligned flash in Internal Trusted Storage

Author
Minos Galanakis

Organization
Arm Limited

Contact
Minos Galanakis <minos.galanakis@arm.com>

Abstract

The proposal is describing a mechanism to enable the use of larger flash devices, imposing a requirement for word-aligned full-block program operations, in Trusted Firmware-M.

Requirements

- Allow page-aligned writes for up to 512 Bytes per page.
- Guarantee data integrity and power-failure reliability.
- Do not alter existing supported platform behaviour.

Current implementation

In the current ITS filesystem design, each filesystem create or write operation requires two flash blocks to be updated: first the data block and then the metadata block. Buffering is avoided as much as possible to reduce RAM requirements.

However, if the ITS_FLASH_PROGRAM_UNIT is 512 Bytes then the data will have to stored in a temporary memory location in order to be able to write that much data in one-shot.

Proposed implementation overview

1. A new block-sized static buffer should be added to its_flash.c when ITS_FLASH_PROGRAM_UNIT is larger than currently supported.
2. Methods calling the flash API such as its_flash_write() or its_flash_block_to_block_move() will populate the buffer instead of directly programming the flash.
3. A new method its_flash_flush(), should be provided in order to flush the block buffer to the device.
4. its_flash_flush() should be called twice: Once after a data block update and once more after the metadata block update is completed.
5. The proposed design should require that the data block update is always completed before the metadata block update starts
6. Writes to the block buffer should be atomic, and guarded against corruption by data from different blocks.

Considerations

- The proposed implementation will increase the RAM usage of ITS by the size of a block, only for platforms which require block-aligned writes.
- Currently power-failure is detected by software by incrementing an 8-bit metadata header field (swap_count), as the last written byte. When the proposed block-buffer is used, the block is programmed in one-shot and the order the bytes are written on the physical device, is hardware dependent.
- A set of guarantees are required by the supported flash ECC devices. The device's flash APIs should provide a mechanism to capture and raise incomplete program operations, as well as write bytes in a sequential order.

For example, if a board powers down through a 512 page program operation, the next read operation should return an error rather than read invalid data.

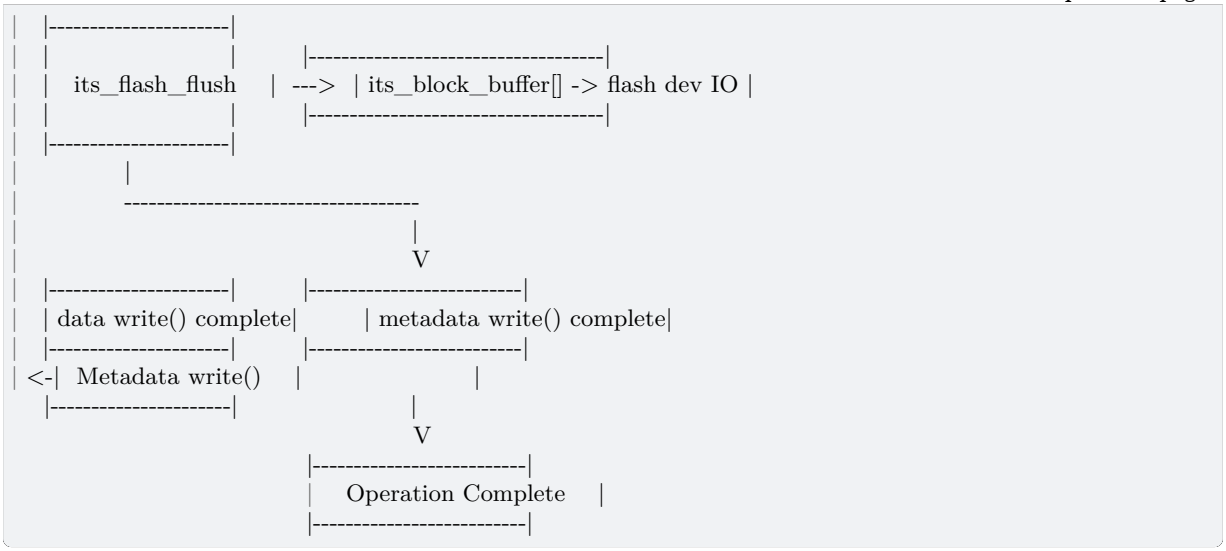
Functional flow diagram

The logic of the proposal is described in the following diagram

```
   |----------------------|
   |  data write()        |
   |----------------------|
   |                      |        |------------------------------|
|-> |  its_flash_write  |     --->  | data[] -> its_block_buffer[] |
|   |                      |        |------------------------------|
```

(continues on next page)

(continued from previous page)



Copyright (c) 2019-2020, Arm Limited. All rights reserved.

PSA Internal Trusted Storage

PSA Internal Trusted Storage (ITS) is a PSA RoT Service for storing the most security-critical device data (e.g. cryptographic keys) in internal storage, which is trusted to provide data confidentiality and authenticity. This contrasts with PSA Protected Storage, which is an Application RoT service that allows larger data sets to be stored securely in external flash, with the option for encryption, authentication and rollback protection to protect the data-at-rest.

Current TF-M Secure Storage

Currently, the TF-M Secure Storage service implements PSA Protected Storage version 1.0-beta2. There is not yet an implementation of PSA Internal Trusted Storage in TF-M.

New TF-M service

The proposal is to implement the *PSA Internal Trusted Storage API* with the *TF-M Internal Trusted Storage service*. It can be abbreviated to *TF-M ITS service* in general and to `its` in code. This name has the advantage of making clear the correspondence between the service and the API it implements.

If this name is adopted, then it may make sense to rename the *Secure Storage service* to the *Protected Storage service* in the future to match. Then "secure storage" could refer to the two services as a collective.

The TF-M ITS service will implement PSA ITS version 1.0. It will be provided by a separate partition to Protected Storage, for a couple of reasons:

- To permit isolation between the services.
  - ITS is a PSA RoT Service, while Protected Storage is an Application RoT Service.
- To avoid circular dependencies.
  - The PSA Firmware Framework does not permit circular dependencies between partitions, which would occur if Protected Storage and ITS were provided by the same partition. Protected Storage depends on Crypto, which in turn depends on ITS.

The existing SST filesystem will be reused to provide the backend of the service, with the flash layer modified to direct storage to internal flash, rather than external.

Compared to Protected Storage, encryption, authentication and rollback protection are not required, so the SST encrypted object layer and the crypto and NV counter interfaces are not required. The rollback protection feature of the object table is also not required.

Code structure

The code structure of the service will be as follows:

TF-M repo:

interface/

- include/psa/internal_trusted_storage.h - PSA ITS API
- src/tfm_its_api.c - PSA ITS API implementation for NSPE

secure_fw/ns_callable/tfm_veneers.c - ITS veneers (auto-generated from manifest)

secure_fw/partitions/internal_trusted_storage/

- tfm_internal_trusted_storage.yaml - Partition manifest
- tfm_its_secure_api.c - PSA ITS API implementation for SPE
- tfm_its_req_mngr.c - Uniform secure functions and IPC request handlers
- tfm_internal_trusted_storage.h - TF-M ITS API (with client_id parameter)
- tfm_internal_trusted_storage.c - TF-M ITS implementation, using the flash_fs as a backend
- its_crypto_interface.h - APIs for encrypting ITS assets used by ITS implementation (optional)
- its_crypto_interface.c - Implementation for ITS encryption (optional)
- platform/ext/target/.../tfm_hal_its_encryption.c - Platform implementation for ITS encryption HAL APIs (optional)
- flash_fs/ - Filesystem
- flash/ - Flash interface

tf-m-tests repo:

test/secure_fw/suites/its/

- non_secure/psa_its_ns_interface_testsuite.c - Non-secure interface tests
- secure/psa_its_s_interface_testsuite.c - Secure interface tests

TF-M ITS implementation

The following APIs will be exposed by tfm_internal_trusted_storage.h:

```
psa_status_t tfm_its_init(void);

psa_status_t tfm_its_set(int32_t client_id,
                         psa_storage_uid_t uid,
                         size_t data_length,
                         const void *p_data,
                         psa_storage_create_flags_t create_flags);

psa_status_t tfm_its_get(int32_t client_id,
                         psa_storage_uid_t uid,
                         size_t data_offset,
                         size_t data_size,
                         void *p_data,
                         size_t *p_data_length);

psa_status_t tfm_its_get_info(int32_t client_id,
```

(continues on next page)

(continued from previous page)

```
                          psa_storage_uid_t uid,
                          struct psa_storage_info_t *p_info);

psa_status_t tfm_its_remove(int32_t client_id,
                          psa_storage_uid_t uid);
```

That is, the TF-M ITS APIs will have the same prototypes as the PSA ITS APIs, but with the addition of a client_id parameter, which will be passed from the ITS request manager. A tfm_its_init function will also be present, which will be called at initialisation time and not exposed through a veneer or SID.

The implementation in tfm_internal_trusted_storage.c must validate the parameters (excepting memory references, which are validated by the SPM), translate the UID and client ID into a file ID and then make appropriate calls to the filesystem layer. It must also take care ensure that any PSA Storage flags associated with the UID are honoured.

Filesystem

The ITS filesystem will be copied and modified from the SST filesystem. The modifications required will be to rename symbols from sst to its and to update the implementation to be aligned with the latest version of the PSA Storage spec (which consists mainly of moving to the psa_status_t error type and using common error codes from psa/error.h).

The filesystem will also be modified to align the size of each file stored to the alignment requirement exposed by the flash interface, by adding appropriate padding.

The filesystem code will be de-duplicated again once the ITS service is implemented (see below).

Flash layer

The flash layer will be copied from SST, and modified to direct writes to the internal flash device. It too needs to be updated to use psa_status_t error types.

Platform layer

The TF-M platform layer must be be updated to distinguish between the external flash device used for Protected Storage and internal flash device used for ITS. A flash region for the relevant storage service needs to be allocated in each.

On test platforms these may just be two distinct regions of the same flash device, but in general they will separate devices with their own drivers.

Detailed design considerations

Mapping UID onto file ID

The ITS APIs identify assets with 64-bit UIDs, to which the ITS service must append the 32-bit client ID of the calling partition for access control. The existing filesystem uses 32-bit file IDs to identify files, so some mapping would be required to convert between the identifiers.

SST uses the object table to do the mapping from client ID, UID pairs to file IDs, which means making an extra filesystem read/write for each get/set operation. This mapping has minimal overhead for SST though, because object table lookups are already required for rollback protection.

For ITS, no rollback protection feature is required, so there are two options:

- Keep a simplified version of the SST object table that just maps from (client ID, UID) to file ID

- Modify the filesystem to take (at least) 96-bit file IDs, in the form of a fixed-length char buffer.

The advantage of the former is that it would require no extra modification to the existing filesystem code, and the existing SST object table could be cut down for ITS. However, it would mean that every ITS request would invoke twice the number of filesystem operations, increasing latency and flash wear. The code size of the ITS partition would be increased, as would RAM usage as the table would need to be read into RAM.

The latter option would make the filesystem slightly more complex: the size of a metadata entry would be increased by 64-bits and the 96-bit fids would need to be copied and compared with memcpy and memcmp calls. On the other hand, mapping onto file IDs would incur only the cost of copying the UID and client ID values into the file ID buffer.

A third, even more general, solution would be to use arbitrary-length null-terminated strings as the file IDs. This is the standard solution in full-featured filesystems, but we do not currently require this level of complexity in secure storage.

With this in mind, the proposed option is the second.

Storing create flags

The ITS APIs provide a 32-bit create_flags parameter, which contains bit flags that determine the properties of the stored data. Only one flag is currently defined for ITS: PSA_STORAGE_FLAG_WRITE_ONCE, which prevents a UID from being modified or deleted after it is set for the first time.

There are two places that these flags could be stored: in the file data or as part of the file metadata.

For the first option, the ITS implementation would need to copy to the flags into the buffer containing the data, and adjust the size accordingly, for each set operation, and the reverse for each get. Every get_info operation would need to read some of the file data, rather than just the metadata, implying a second flash read. A potential downside is that many of the cryptographic assets stored in ITS will be aligned to power-of-two sizes; adding an extra 32-bits would misalign the size, which may reduce flash performance or necessitate adding padding to align to the flash page size.

To implement the second option, a 32-bit flag field would be added to the filesystem's metadata structure, whose interpretation is defined by the user. This field would clearly be catered towards the PSA Storage APIs, even if nominally generic, and alternative filesystems may not have any such field. However, it is a more intuitive solution and would simplify both flash alignment and get_info operations.

Overall, it seems more beneficial to store the flags in the metadata, so this is the proposed solution.

Code sharing between Protected Storage and ITS

To de-duplicate the filesystem code used by both Protected Storage and ITS, it is proposed that Protected Storage calls ITS APIs as its backend filesystem.

Protected Storage essentially becomes an encryption, authentication and rollback protection layer on top of ITS. It makes IPC requests or secure function calls to the ITS service to do filesystem operations on its behalf.

This has a couple of advantages:

- It shrinks Protected Storage's stack size, because the filesystem and flash layer stack is only in ITS.
- It automatically solves the problem of ensuring mutual exclusion in the filesystem and flash layers when Protected Storage and ITS are called concurrently. The second request to ITS will just be made to wait by the SPM.

The disadvantage of this approach is that it will increase the latency of Protected Storage requests, due to the extra overhead associated with making a second IPC request or secure function call. It also limits Protected Storage to using only the ITS APIs, unless extra veneers are added solely for Protected Storage to use. This, for example, prevents Protected Storage from doing partial writes to file without reading and re-writing the whole file.

ITS will need to be modified to direct calls from Protected Storage to a different flash device. It can use the client ID to detect when the caller is Protected Storage, and pass down the identity of the flash device to use to the flash layer, which then calls the appropriate driver.

An open question is what to do if Protected Storage itself wants to store something in internal storage in the future (e.g. rollback counters, hash tree/table or top hash). A couple of possible solutions would be:

- Divide up the UIDs, so certain UIDs from Protected Storage refer to assets in internal storage, and others to ones in external storage.
- Use the type field of psa_call in IPC model to distinguish between internal and external storage requests.

The other option for code sharing would be for Protected Storage and ITS to directly share filesystem code, which would be placed in a shared code region. With this approach, mutual exclusion to the flash device would need to be implemented separately, as would some way of isolating static memory belonging to each partition but not the code. Because of these complications, this option has not been considered further at this time.

Encryption in ITS

The ITS can optionally be configured to encrypt the internal trusted storage data. To support encryption in ITS the target platform must provide an implementation of the APIs defined in platform/include/tfm_hal_its_encryption.h:

```
enum tfm_hal_status_t tfm_hal_its_aead_generate_nonce(uint8_t *nonce,
                                                      const size_t nonce_size);

enum tfm_hal_status_t tfm_hal_its_aead_encrypt(
                                struct tfm_hal_its_auth_crypt_ctx *ctx,
                                const uint8_t *plaintext,
                                const size_t plaintext_size,
                                uint8_t *ciphertext,
                                const size_t ciphertext_size,
                                uint8_t *tag,
                                const size_t tag_size);

enum tfm_hal_status_t tfm_hal_its_aead_decrypt(
                                struct tfm_hal_its_auth_crypt_ctx *ctx,
                                const uint8_t *ciphertext,
                                const size_t ciphertext_size,
                                uint8_t *tag,
                                const size_t tag_size,
                                uint8_t *plaintext,
                                const size_t plaintext_size);
```

Then encryption can be enabled by setting the build option -DITS_ENCRYPTION=ON.

The figure fig-tfm_eits describes the encryption and decryption process happening when calling tfm_its_set and tfm_its_get.

By using an AEAD scheme, it is possible to not only encrypt the file data but also authenticate the file meta data, which include:

- File id
- File size

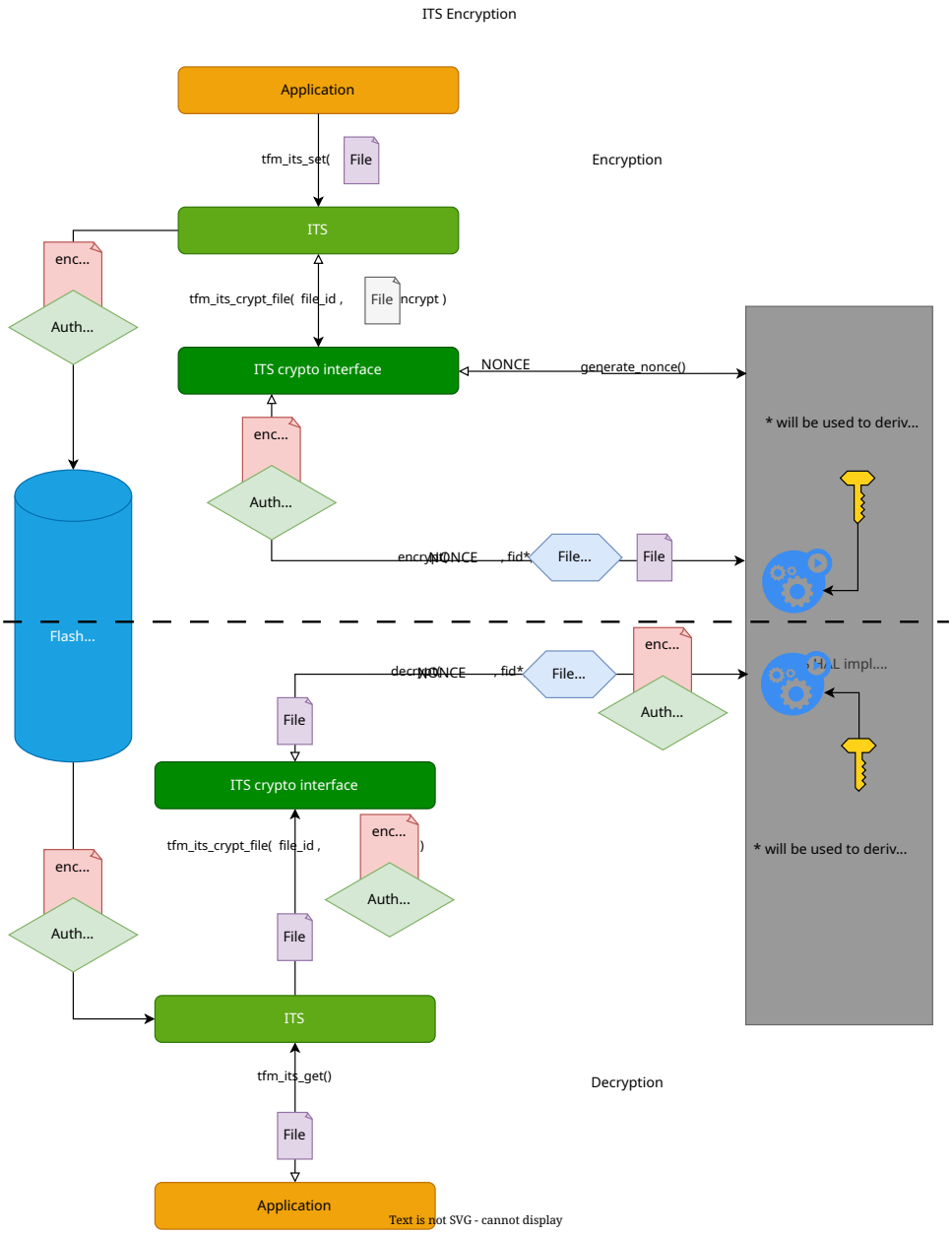


Fig. 18: En/Decryption of ITS

- File flags

The key used to perform the AEAD operation must be derived from a long-term key-derivation key and the file id, which is used as a derivation label. The long-term key-derivation key must be managed by the target platform.

There is a generic implementation of the abovementioned functions under platform/ext/common/template/tfm_hal_its_encryption.c using PSA crypto calls similar to Protected Storage solution. When used, the default NV seed template under platform/ext/common/template/crypto_nv_seed.c must be disabled, as it relies on ITS. If there is a need for NV seed usage, an ITS independent implementation is required. If NV seed is not necessary, it can be turned off by setting CRYPTO_NV_SEED=0.

Copyright (c) 2019-2024, Arm Limited. All rights reserved.

Firmware Update Service

Author
Sherry Zhang

Organization
Arm Limited

Contact
Sherry Zhang <Sherry.Zhang2@arm.com>

Table of Contents

- *Firmware Update Service*
  - *Introduction of Firmware Update service*
  - *Components*
  - *Service API description*
  - *Shim Layer between FWU and bootloader*
    - *Shim layer introduction*
    - *Interfaces of the shim Layer*
  - *Additional shared data between BL2 and SPE*
  - *Build configurations related to FWU partition*
  - *Limitations of current implementation*
  - *Benefits Analysis on this Partition*
    - *Implement the FWU functionality in the non-secure side*
    - *Pros and Cons for implementing FWU APIs in secure side*
  - *Reference*

Introduction of Firmware Update service

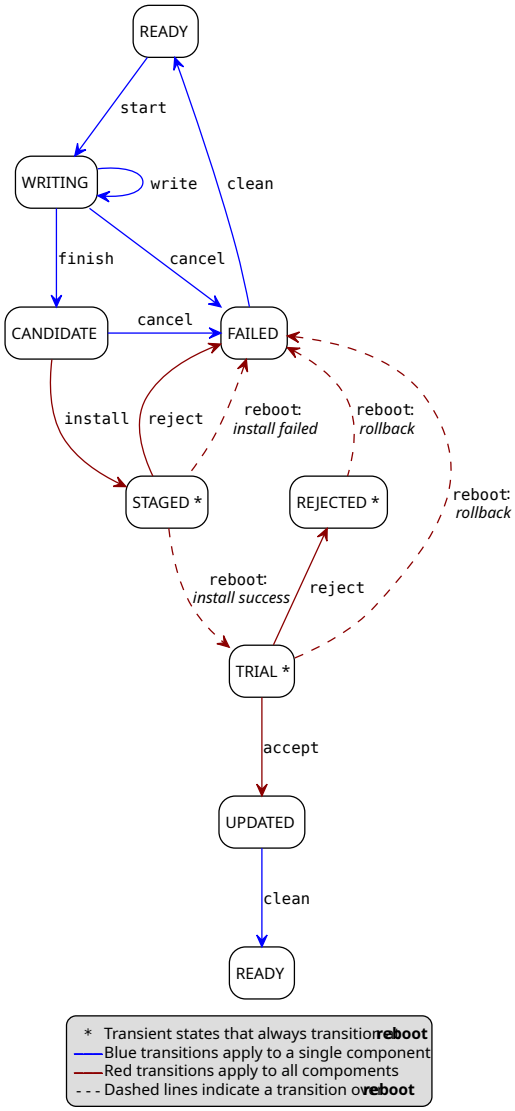
The Firmware Update(FWU) service provides the functionality of updating firmware images. It provides a standard interface for updating firmware and it is platform independent. TF-M defines a shim layer to support cooperation between bootloader and FWU service.

This partition supports the following features:

- Query the firmware store information.

- Image preparation: prepare a new firmware image in the component's firmware store.
- Image installation: install prepared firmware images on all components that have been prepared for installation.
- Image trial: manage a trial of new firmware images atomically on all components that are in TRIAL state.

A typical flow through the component states is shown below[1].



Components

The structure of the TF-M Firmware Update service is listed below:

[1] PSA Firwmare Update API

Component name	Description	Location
Client API interface	This module exports the client API of PSA Firmware Update to the users.	./interface/src/tfm_fwu_api.c
Manifest	The manifest file is a description of the service components.	./secure_fw/partitions/firmware_update/tfm_firmware_update.yaml
NSPE client API interface	This module exports the client API of PSA Firmware Update to the NSPE(i.e. to the applications).	./interface/src/tfm_fwu_api.c
IPC request handlers	This module handles all the secure requests in IPC model. It maitains the image state context and calls the image ID converter to achieve the firmware update functionalities.	./secure_fw/partitions/firmware_update/tfm_fwu_req_mngr.c
Shim layer between FWU and bootloader	This module provides the APIs with the functionality of operating the bootloader to cooperate with the Firmware Update service	./secure_fw/partitions/firmware_update/bootloader/tfm_bootloader_fwu_abstraction.h
Shim layer example based on MCUboot	This module is the implementation of the shim layer between FWU and bootloader based on MCUboot.	./secure_fw/partitions/firmware_update/bootloader/mcuboot/tfm_mcuboot_fwu.c

Service API description

This service follows the PSA Firmware Update API spec of version 1.0[Page 1436, 1]. Please refer to Firmware Update spec for the detailed description.

Shim Layer between FWU and bootloader

The firmware update operations are achieved by calling the shim layer APIs between bootloader and FWU.

Shim layer introduction

This shim layer provides the APIs with the functionality of operating the bootloader to cooperate with the Firmware Update service. This shim layer is decoupled from bootloader implementation. Users can specify a specific bootloader by setting TFM_FWU_BOOTLOADER_LIB build configuration and adding the specific build scripts into that file. By default, the MCUboot is chosen as the bootloader.

Interfaces of the shim Layer

fwu_bootloader_init(function)

Prototype

```
psa_status_t fwu_bootloader_init(void);
```

Description

Bootloader related initialization for the firmware update. It reads some necessary shared data from the memory if needed. It initializes the flash drivers defined in FLASH_DRIVER_LIST. Platform can define FLASH_DRIVER_LIST in flash_layout.h to overload the default driver list.

Parameters

N/A

fwu_bootloader_staging_area_init(function)

Prototype

```
psa_status_t fwu_bootloader_staging_area_init(psa_fwu_component_t component,
                                              const void *manifest,
                                              size_t manifest_size);
```

Description

The component is in READY state. Prepare the staging area of the component for image download. For example, initialize the staging area, open the flash area, and so on.

Parameters

- component: The identifier of the target component in bootloader.
- manifest: A pointer to a buffer containing a detached manifest for the update. If the manifest is bundled with the firmware image, manifest must be NULL.
- manifest_size: Size of the manifest buffer in bytes.

fwu_bootloader_load_image(function)

Prototype

```
psa_status_t fwu_bootloader_load_image(psa_fwu_component_t component,
                                       size_t       image_offset,
                                       const void    *block,
                                       size_t       block_size);
```

Description

Load the image into the target component.

Parameters

- component: The identifier of the target component in bootloader.
- image_offset: The offset of the image being passed into block, in bytes.
- block: A buffer containing a block of image data. This might be a complete image or a subset.
- block_size: Size of block.

fwu_bootloader_install_image(function)

Prototype

```
psa_status_t fwu_bootloader_install_image(psa_fwu_component_t *candidates,
                                          uint8_t number);
```

Description

Check the authenticity and integrity of the image. If a reboot is required to complete the check, then mark this image as a candidate so that the next time bootloader runs it will take this image as a candidate one to bootup. Return the error code PSA_SUCCESS_REBOOT.

Parameters

- candidates: A list of components in CANDIDATE state.
- number: Number of components in CANDIDATE state.

fwu_bootloader_mark_image_accepted(function)

Prototype

```
psa_status_t fwu_bootloader_mark_image_accepted(const psa_fwu_component_t *trials,
                                                uint8_t number);
```

Description

Call this API to mark the TRIAL(running) image in component as confirmed to avoid revert when next time bootup. Usually, this API is called after the running images have been verified as valid.

Parameters

- trials: A list of components in TRIAL state.
- number: Number of components in TRIAL state.

fwu_bootloader_reject_staged_image(function)

Prototype

```
psa_status_t fwu_bootloader_reject_staged_image(psa_fwu_component_t component);
```

Description

The component is in STAGED state. Call this API to Uninstall the staged image in the component so that this image will not be treated as a candidate next time bootup.

Parameters

- component: The identifier of the target component in bootloader.

fwu_bootloader_reject_trial_image(function)

Prototype

```
psa_status_t fwu_bootloader_reject_trial_image(psa_fwu_component_t component);
```

Description

The component is in TRIAL state. Mark the running image in the component as rejected.

Parameters

- component: The identifier of the target component in bootloader.

fwu_bootloader_clean_component(function)

Prototype

```
psa_status_t fwu_bootloader_clean_component(psa_fwu_component_t component);
```

Description

The component is in FAILED or UPDATED state. Clean the staging area of the component.

Parameters

- component: The identifier of the target component in bootloader.

fwu_bootloader_get_image_info(function)

Prototype

```
psa_status_t fwu_bootloader_get_image_info(psa_fwu_component_t component,
                                           bool query_state,
                                           bool query_impl_info,
                                           psa_fwu_component_info_t *info);
```

Description

Get the image information of the given bootloader_image_id in the staging area or the running area.

Parameters

- component: The identifier of the target component in bootloader.
- query_state: Whether query the 'state' field of psa_fwu_component_info_t.
- query_impl_info: Whether Query 'impl' field of psa_fwu_component_info_t.
- info: Buffer containing return the component information.

Additional shared data between BL2 and SPE

An additional TLV area "image version" is added into the shared memory between BL2 and TF-M. So that the firmware update partition can get the image version. Even though the image version information is also included in the BOOT RECORD TLV area which is encoded by CBOR, adding a dedicated image version TLV area is preferred to avoid involving the CBOR encoder which can increase the code size. The FWU partition will read the shared data at the partition initialization.

Build configurations related to FWU partition

- TFM_PARTITION_FIRMWARE_UPDATE Controls whether FWU partition is enabled or not.
- TFM_FWU_BOOTLOADER_LIB Bootloader configure file for FWU partition.
- TFM_CONFIG_FWU_MAX_WRITE_SIZE The maximum permitted size for block in psa_fwu_write, in bytes.
- TFM_FWU_BUF_SIZE Size of the FWU internal data transfer buffer (defaults to TFM_CONFIG_FWU_MAX_WRITE_SIZE if not set).
- FWU_STACK_SIZE The stack size of FWU Partition.
- FWU_DEVICE_CONFIG_FILE The device configuration file for FWU partition. The default value is the configuration file generated for MCUboot. The following macros should be defined in the configuration file:
  - FWU_COMPONENT_NUMBER The number of components on the device.

    Note: In this design, component ID ranges from 0 to FWU_COMPONENT_NUMBER - 1.

  - FWU_SUPPORT_TRIAL_STATE Whether TRIAL component state is supported.
- TEST_NS_FWU FWU nonsecure tests switch.
- TEST_S_FWU FWU secure tests switch.

Note: The running image which supports revert mechanism should be confirmed before initiating a firmware update process. For example, if the running image is built with -DMCUBOOT_UPGRADE_STRATEGY=SWAP_USING_MOVE, the image should be confirmed either by adding -DMCUBOOT_CONFIRM_IMAGE=ON build option or by calling psa_fwu_accept() API before initiating a firmware

update process. Otherwise, PSA_ERROR_BAD_STATE will be returned by psa_fwu_start().

Limitations of current implementation

Currently, the MCUboot based implementation does not record image update results like failure or success. And FWU partition does not detect failure errors in bootloader installation. If an image installation fails in the bootloader and the old image still runs after reboot, PSA_FWU_READY state will be returned by psa_fwu_query() after reboot.

Currently, image download recovery after a reboot is not supported. If a reboot happens in image preparation, the downloaded image data will be ignored after the reboot.

Benefits Analysis on this Partition

Implement the FWU functionality in the non-secure side

The APIs listed in PSA Firmware Update API spec[Page 1436, 1] can also be implemented in the non-secure side.

Pros and Cons for implementing FWU APIs in secure side

Pros

- It protects the image in the passive or staging area from being tampered with by the NSPE. Otherwise, a malicious actor from NSPE can tamper the image stored in the non-secure area to break image update.
- It protects secure image information from disclosure. In some cases, the non-secure side shall not be permitted to get secure image information.
- It protects the active image from being manipulated by NSPE. Some bootloader supports testing the image. After the image is successfully installed and starts to run, the user should set the image as permanent image if the image passes the test. To achieve this, the area of the active image needs to be accessed. In this case, implementing FWU service in SPE can prevent NSPE from manipulating the active image area.
- On some devices, such as the Arm Musca-B1 board, the passive or staging area is restricted as secure access only. In this case, the FWU partition should be implemented in the secure side.

Cons

- It increases the image size of the secure image.
- It increases the execution latency and footprint. Compared to implementing FWU in NSPE directly, calling the Firmware Update APIs which are implemented in the secure side increases the execution latency and footprint.
- It can increase the attack surface of the secure runtime.

Users can decide whether to call the FWU service in TF-M directly or implement the Firmware Update APIs in the non-secure side based on the pros and cons analysis above.

Reference

Copyright (c) 2021-2022, Arm Limited. All rights reserved.

Protected Storage service key management

Author
Jamie Fox

Organization
Arm Limited

Contact
Jamie Fox <jamie.fox@arm.com>

Background

The PSA Protected Storage API requires confidentiality for external storage to be provided by:

> **cryptographic ciphers using device-bound keys**, a tamper resistant enclosure, or an inaccessible deployment location, depending on the threat model of the deployed system.

A TBSA-M-compliant device must embed a Hardware Unique Key (HUK), which provides the root of trust (RoT) for confidentiality in the system. It must have at least 128 bits of entropy (and a 128 bit data size), and be accessible only to Trusted code or Trusted hardware that acts on behalf of Trusted code. [TBSA-M]

Design description

Each time the system boots, PS will request that the Crypto service uses a key derivation function (KDF) to derive a storage key from the HUK, by referring to the builtin key handle for the HUK. The storage key could be kept in on-chip volatile memory private to the Crypto partition, or it could remain inside a secure element. Either way it will not be returned to PS.

For each call to the PSA Protected Storage APIs, PS will make requests to the Crypto service to perform AEAD encryption and/or decryption operations using the storage key (providing a fresh nonce for each encryption).

At no point will PS access the key material itself, only referring to the HUK and storage key by their handles in the Crypto service.

Key derivation

PS will make key derivation requests to the Crypto service with calls to the PSA Crypto APIs. In order to derive the storage key, the following calls are required:

```
status = psa_key_derivation_setup(&op, PSA_ALG_HKDF(PSA_ALG_SHA_256));

/* Set up a key derivation operation with HUK  */
status = psa_key_derivation_input_key(&op, PSA_KEY_DERIVATION_INPUT_SECRET,
                                      TFM_BUILTIN_KEY_ID_HUK);

/* Supply the PS key label as an input to the key derivation */
status = psa_key_derivation_input_bytes(&op, PSA_KEY_DERIVATION_INPUT_INFO,
                                        key_label,
                                        key_label_len);

/* Create the storage key from the key derivation operation */
status = psa_key_derivation_output_key(&attributes, &op, &ps_key);
```

Note: TFM_BUILTIN_KEY_ID_HUK is a static key ID that is used to identify the HUK. It has an arbitrary value defined in tfm_crypto_defs.h

ps_key is a PSA Crypto key handle to a volatile key, set by the derivation operation. After the call to psa_key_derivation_output_key, it can be used to refer the storage key.

key_label can be any string that is independent of the input key material and different to the label used in any other derivation from the same input key. It prevents two different contexts from deriving the same output key from the same input key.

The key derivation function used by the crypto service to derive the storage key will be HKDF, with SHA-256 as the underlying hash function. HKDF is suitable because:

- It is simple and efficient, requiring only two HMAC operations when the length of the output key material is less than or equal to the hash length (as is the case here).
- The trade-off is that HKDF is only suitable when the input key material has at least as much entropy as required for the output key material. But this is the case here, as the HUK has 128 bits of entropy, the same as required by PS.
- HKDF is standardised in RFC 5869 [RFC5869] and its security has been formally analysed. [HKDF]
- It is supported by the TF-M Crypto service.

The choice of underlying hash function is fairly straightforward: it needs to be a modern standardised algorithm, considered to be secure and supported by TF-M Crypto. This narrows it down to just the SHA-2 family. Of the hash functions in the family, SHA-256 is the simplest and provides more than enough output length.

Keeping the storage key private to PS

The Crypto service derives a platform key from the HUK, using the partition ID as the input to that derivation, and that platform key is used for the key derivation by PS. This happens transparently, and to PS is indistinguishable from deriving from the HUK except that other partitions cannot derive the storage key even if they know the derivation parameters.

Key use

To encrypt and decrypt data, PS will call the PSA Crypto AEAD APIs in the same way as the current implementation, but ps_key will refer to the storage key, rather than the imported HUK. For each encryption operation, the following call is made (and analogously for decryption):

```
psa_aead_encrypt(ps_key, PS_CRYPTO_ALG,
                 crypto->ref.iv, PS_IV_LEN_BYTES,
                 add, add_len,
                 in, in_len,
                 out, out_size, out_len);
```

References

Copyright (c) 2019-2022, Arm Limited. All rights reserved.

Copyright (c) 2023-2024, Arm Limited. All rights reserved.

Software Design Notes

Code sharing between independently linked XIP binaries

Author
Tamas Ban

Organization
Arm Limited

Contact
tamas.ban@arm.com

Motivation

Cortex-M devices are usually constrained in terms of flash and RAM. Therefore, it is often challenging to fit bigger projects in the available memory. The PSA specifications require a device to both have a secure boot process in place at device boot-up time, and to have a partition in the SPE which provides cryptographic services at runtime. These two entities have some overlapping functionality. Some cryptographic primitives (e.g. hash calculation and digital signature verification) are required both in the bootloader and the runtime environment. In the current TF-M code base, both firmware components use the mbed-crypto library to implement these requirements. During the build process, the mbed-crpyto library is built twice, with different configurations (the bootloader requires less functionality) and then linked to the corresponding firmware component. As a result of this workflow, the same code is placed in the flash twice. For example, the code for the SHA-256 algorithm is included in MCUboot, but the exact same code is duplicated in the SPE cryptography partition. In most cases, there is no memory isolation between the bootloader and the SPE, because both are part of the PRoT code and run in the secure domain. So, in theory, the code of the common cryptographic algorithms could be reused among these firmware components. This could result in a big reduction in code footprint, because the cryptographic algorithms are usually flash hungry. Code size reduction can be a good opportunity for very constrained devices, which might need to use TF-M Profile Small anyway.

Technical challenge

Code sharing in a regular OS environment is easily achievable with dynamically linked libraries. However, this is not the case in Cortex-M systems where applications might run bare-metal, or on top of an RTOS, which usually lacks dynamic loading functionality. One major challenge to be solved in the Cortex-M space is how to share code between independently linked XIP applications that are tied to a certain memory address range to be executable and have absolute function and global data memory addresses. In this case, the code is not relocatable, and in most cases, there is no loader functionality in the system that can perform code relocation. Also, the lack of an MMU makes the address space flat, constant and not reconfigurable at runtime by privileged code.

One other difficulty is that the bootloader and the runtime use the same RAM area during execution. The runtime firmware is executed strictly after the bootloader, so normally, it can reuse the whole secure RAM area, as it would be the exclusive user. No attention needs to be paid as to where global data is placed by the linker. The bootloader does not need to retain its state. The low level startup of the runtime firmware can freely overwrite the RAM with its data without corrupting bootloader functionality. However, with code sharing between bootloader and runtime firmware, these statements are no longer true. Global variables used by the shared code must either retain their value or must be reinitialised during low level startup of the runtime firmware. The startup code is not allowed to overwrite the shared global variables with arbitrary data. The following design proposal provides a solution to these challenges.

Design concept

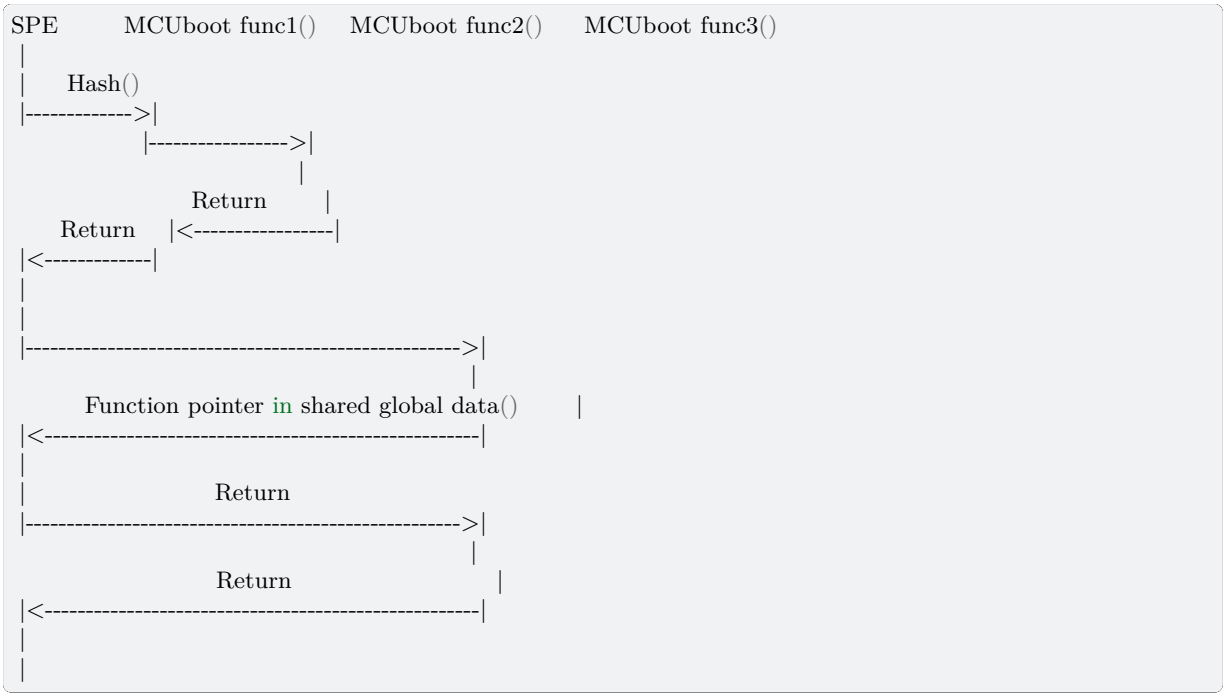
The bootloader is sometimes implemented as ROM code (BL1) or stored in a region of the flash which is lockable, to prevent tampering. In a secure system, the bootloader is immutable code and thus implements a part of the Root of Trust anchor in the device, which is trusted implicitly. The shared code is primarily part of the bootloader, and is reused by the runtime SPE firmware at a later stage. Not all of the bootloader code is reused by the runtime SPE, only some cryptographic functions.

Simplified steps of building with code sharing enabled:

- Complete the bootloader build process to have a final image that contains the absolute addresses of the shared functions, and the global variables used by these functions.

- Extract the addresses of the functions and related global variables that are intended to be shared from the bootloader executable.
- When building runtime firmware, provide the absolute addresses of the shared symbols to the linker, so that it can pick them up, instead of instantiating them again.

The execution flow looks like this:

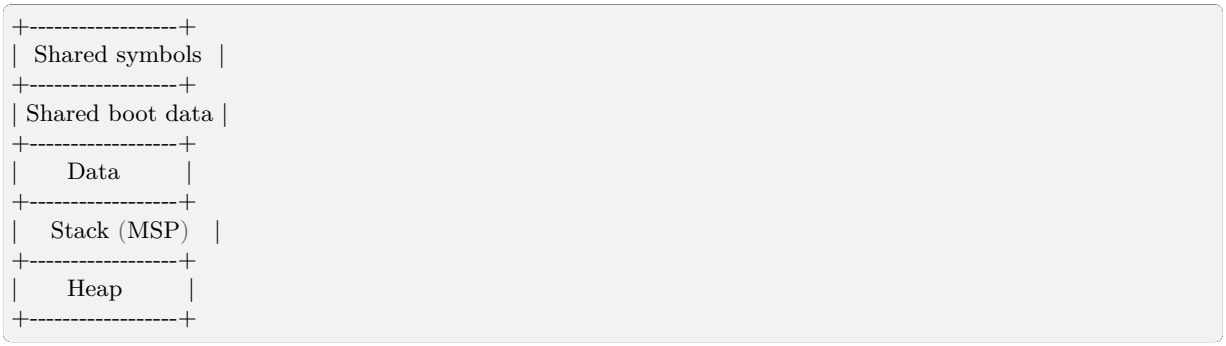


The execution flow usually returns from a shared function back to the SPE with an ordinary function return. So usually, once a shared function is called in the call path, all further functions in the call chain will be shared as well. However, this is not always the case, as it is possible for a shared function to call a non-shared function in SPE code through a global function pointer.

For shared global variables, a dedicated data section must be allocated in the linker configuration file. This area must have the same memory address in both MCUboot's and the SPE's linker files, to ensure the integrity of the variables. For simplicity's sake, this section is placed at the very beginning of the RAM area. Also, the RAM wiping functionality at the end of the secure boot flow (that is intended to remove any possible secrets from the RAM) must not clear this area. Furthermore, it must be ensured that the linker places shared globals into this data section. There are two way to achieve this:

- Put a filter pattern in the section body that matches the shared global variables.
- Mark the global variables in the source code with special attribute *__attribute__((section(<NAME_OF_SHARED_SYMBOL_SECTION>)))*

RAM memory layout in MCUboot with code sharing enabled:



RAM memory layout in SPE with code sharing enabled:

```
+------------------+
|  Shared symbols  |
+------------------+
| Shared boot data |
+------------------+
|   Stack (MSP)    |
+------------------+
|   Stack (PSP)    |
+------------------+
| Partition X Data |
+------------------+
| Partition X Stack|
+------------------+
         .
         .
         .
+------------------+
| Partition Z Data |
+------------------+
| Partition Z Stack|
+------------------+
|    PRoT Data     |
+------------------+
|      Heap        |
+------------------+
```

Patching Mbed TLS

In order to share some global function pointers from mbed-crypto that are related to dynamic memory allocation, their scope must be extended from private to global. This is needed because some compiler toolchain only extract the addresses of public functions and global variables, and extraction of addresses is a requirement to share them among binaries. Therefore, a short patch was created for the mbed-crypto library, which "globalises" these function pointers:

lib/ext/mbedcrypto/0002-Enable-crypto-code-sharing-between-independent-binar.patch

The patch needs to be manually applied in the Mbed TLS repo, if code sharing is enabled. The patch has no effect on the functional behaviour of the cryptographic library, it only extends the scope of some variables.

Tools support

All the currently supported compilers provide a way to achieve the above objectives. However, there is no standard way, which means that the code sharing functionality must be implemented on a per compiler basis. The following steps are needed:

- Extraction of the addresses of all global symbols.
- The filtering out of the addresses of symbols that aren't shared. The goal is to not need to list all the shared symbols by name. Only a simple pattern has to be provided, which matches the beginning of the symbol's name. Matching symbols will be shared. Examples are in : *bl2/shared_symbol_template.txt*
- Provision of the addresses of shared symbols to the linker during the SPE build process.
- The resolution of symbol collisions during SPE linking. Because mbed-crypto is linked to both firmware components as a static library, the external shared symbols will conflict with the same symbols found within it. In order to prioritize the external symbol, the symbol with the same name in mbed-crypto must be marked as weak in the symbol table.

The above functionalities are implemented in the toolchain specific CMake files:

- *toolchain_ARMCLANG.cmake*

- *toolchain_GNUARM.cmake*

By the following two functions:

- *target_share_symbols()*: Extract and filter shared symbol addresses from MCUboot.
- *target_link_shared_code()*: Link shared code to the SPE and resolve symbol conflict issues.

ARMCLANG

The toolchain specific steps are:

- Extract all symbols from MCUboot: add *-symdefs* to the compiler command line
- Filter shared symbols: call CMake script *FilterSharedSymbols.cmake*
- Weaken duplicated (shared) symbols in the mbed-crypto static library that are linked to the SPE: *arm-none-eabi-objcopy*
- Link shared code to SPE: Add the filtered output of *-symdefs* to the SPE source file list.

GNUARM

The toolchain specific steps are:

- Extract all symbols from MCUboot: *arm-none-eabi-nm*
- Filter shared symbols: call CMake script: *FilterSharedSymbols.cmake*
- Strip unshared code from MCUboot: *arm-none-eabi-strip*
- Weaken duplicated (shared) symbols in the mbed-crypto static library that are linked to the SPE: *arm-none-eabi-objcopy*
- Link shared code to SPE: Add *-Wl -R <SHARED_STRIPPED_CODE.axf>* to the compiler command line

IAR

Functionality currently not implemented, but the toolchain supports doing it.

Memory footprint reduction

Build type: MinSizeRel Platform: mps2/an521 Version: TF-Mv1.2.0 + code sharing patches MCUboot image encryption support is disabled.

	ConfigDefault		ConfigProfile-M		ConfigProfile-S	
	ARM-CLANG	GNUARM	ARM-CLANG	GNUARM	ARM-CLANG	GNUARM
CODE_SHARING=OFI	122268	124572	75936	75996	50336	50224
CODE_SHARING=ON	113264	115500	70400	70336	48840	48988
Difference	9004	9072	5536	5660	1496	1236

If MCUboot image encryption support is enabled then saving could be up to ~13-15KB.

Note: Code sharing on Musca-B1 was tested only with SW only crypto, so crypto hardware acceleration must be turned off: -DCRYPTO_HW_ACCELERATOR=OFF

Useability considerations

Functions that only use local variables can be shared easily. However, functions that rely on global variables are a bit tricky. They can still be shared, but all global variables must be placed in the shared symbol section, to prevent overwriting and to enable the retention of their values.

Some global variables might need to be reinitialised to their original values by runtime firmware, if they have been used by the bootloader, but need to have their original value when runtime firmware starts to use them. If so, the reinitialising functionality must be implemented explicitly, because the low level startup code in the SPE does not initialise the shared variables, which means they retain their value after MCUboot stops running.

If a bug is discovered in the shared code, it cannot be fixed with a firmware upgrade, if the bootloader code is immutable. If this is the case, disabling code sharing might be a solution, as the new runtime firmware could contain the fixed code instead of relying on the unfixed shared code. However, this would increase code footprint.

API backward compatibility also can be an issue. If the API has changed in newer version of the shared code. Then new code cannot rely on the shared version. The changed code and all the other shared code where it is referenced from must be ignored and the updated version of the functions must be compiled in the SPE binary. The Mbed TLS library is API compatible with its current version (v2.24.0) since the mbedtls-2.7.0 release (2018-02-03).

To minimise the risk of incompatibility, use the same compiler flags to build both firmware components.

The artifacts of the shared code extraction steps must be preserved so as to remain available if new SPE firmware (that relies on shared code) is built and released. Those files are necessary to know the address of shared symbols when linking the SPE.

How to use code sharing?

Considering the above, code sharing is an optional feature, which is disabled by default. It can be enabled from the command line with a compile time switch:

- *TFM_CODE_SHARING*: Set to *ON* to enable code sharing.

With the default settings, only the common part of the mbed-crypto library is shared, between MCUboot and the SPE. However, there might be other device specific code (e.g. device drivers) that could be shared. The shared cryptography code consists mainly of the SHA-256 algorithm, the *bignum* library and some RSA related functions. If image encryption support is enabled in MCUboot, then AES algorithms can be shared as well.

Sharing code between the SPE and an external project is possible, even if MCUboot isn't used as the bootloader. For example, a custom bootloader can also be built in such a way as to create the necessary artifacts to share some of its code with the SPE. The same artifacts must be created like the case of MCUboot:

- ***shared_symbols_name.txt*: Contains the name of the shared symbols. Used by** the script that prevents symbol collision.
- *shared_symbols_address.txt*: Contains the type, address and name of shared symbols. Used by the linker when linking runtime SPE.
- *shared_code.axf*: GNUARM specific. The stripped version of the firmware component, only contains the shared code. It is used by the linker when linking the SPE.

Note: The artifacts of the shared code extraction steps must be preserved to be able to link them to any future SPE version.

When an external project is sharing code with the SPE, the *SHARED_CODE_PATH* compile time switch must be set to the path of the artifacts mentioned above.

Further improvements

This design focuses only on sharing the cryptography code. However, other code could be shared as well. Some possibilities:

- Flash driver
- Serial driver
- Image metadata parsing code
- etc.

Copyright (c) 2020-2024, Arm Limited. All rights reserved.

Hardware Abstraction Layer

Organization
Arm Limited

Contact
tf-m@lists.trustedfirmware.org

API Version
0.9

Introduction

TF-M HAL abstracts the hardware-oriented and platform specific operations on the *SPE* side and provides a set of APIs to the upper layers such as *SPM*, *RoT Service*. The *HAL* aims to cover the platform different aspects whereas common architecturally defined aspects are done generically within the common *SPE*. In some cases, although the operations are defined architecturally, it may not be possible to generalize implementations because lots of information is only known to platforms. It is more efficient to define a *HAL* API for those architectural operations as well.

Note: *TBSA-M* provides the hardware requirements for security purposes. *TF-M HAL* tries to reference *TBSA-M* recommendations in the interfaces from the software perspective only. Please reference *TBSA-M* for your security hardware design.

Design Goals

TF-M HAL is designed to simplify the integration efforts on different platforms.

TF-M HAL is designed to make it easy to use the hardware and develop the *SPM* and *RoT Service* which need to access the devices.

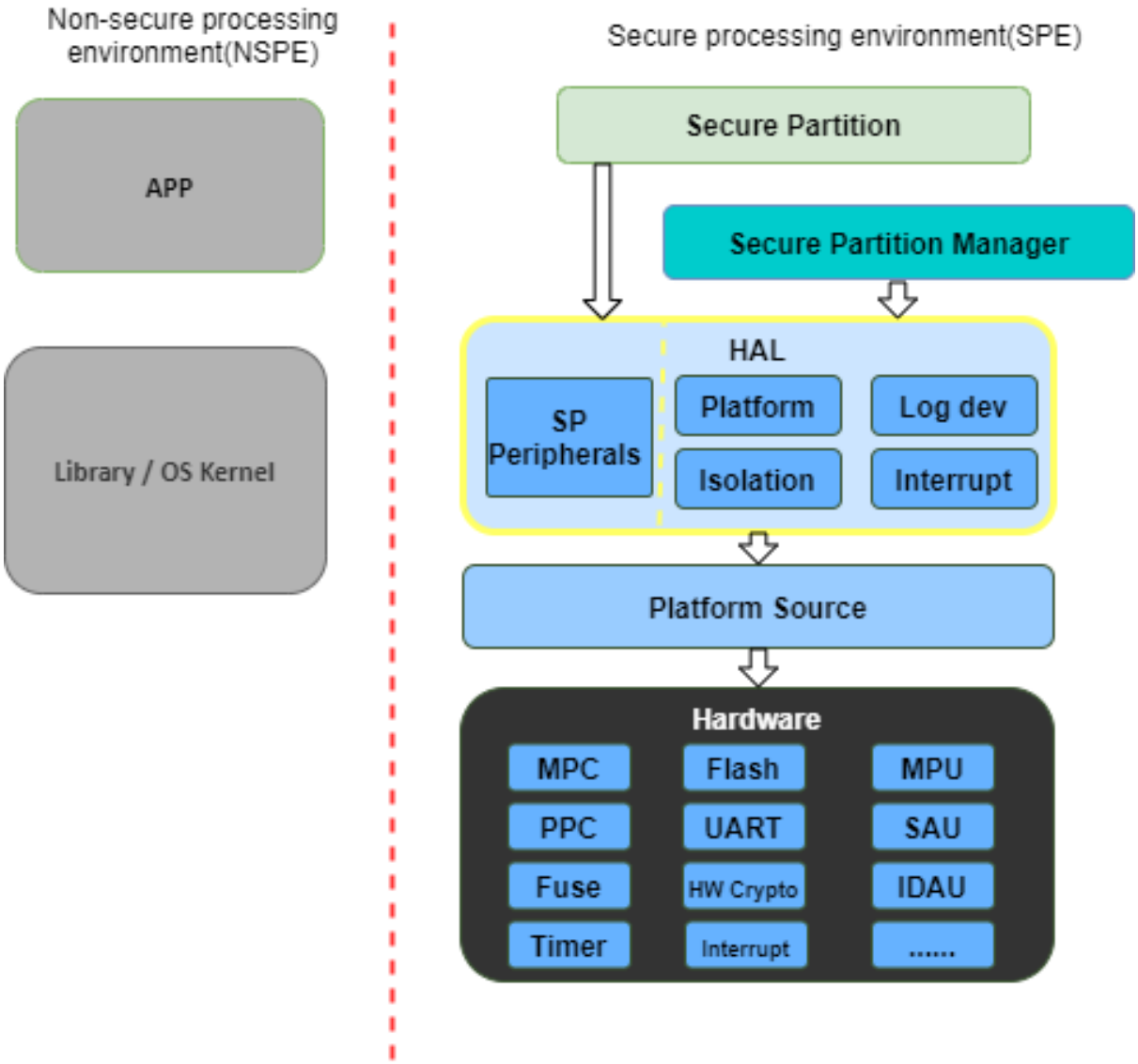
TF-M HAL is designed to make the structure clearer and let the *TF-M* mainly focus on *PSA* implementation.

Overview

This section provides an overview of the abstraction layer structure.

Here lists a minimal set of necessary functionalities:

- **Isolation API**: Provides the necessary isolation functionalities required by the *PSA-FF-M* and *TBSA-M*, and provides APIs to *SPM* to check the permissions of memory access.
- **Platform API**: Provides the platform initialization, platform-specific memory information, system reset, etc.
- **Log dev API**: Provides the log system functions.
- **Interrupt API**: Provides the interrupt functions.


Non-secure processing environment(NSPE)
APP
Library / OS Kernel
Secure processing environment(SPE)
Secure Partition
Secure Partition Manager
HAL
SP Peripherals
Platform
Log dev
Isolation
Interrupt
Platform Source
Hardware
MPC
Flash
MPU
PPC
UART
SAU
Fuse
HW Crypto
IDAU
Timer
Interrupt
.....

Note:

- There is a non-secure *HAL* that focuses on the mailbox operation API for Dual-core topology. For more information about it, please refer to *Mailbox Design in TF-M on Dual-core System*.
- The minimal set of *TF-M HAL* is sufficient for Secure Partitions by using customized peripheral interfaces. To provide easier portability for the Secure Partitions, a Secure Partition *HAL* is provided in this design too.
- The debug mechanisms give the external entity the corresponding right to access the system assets. *TF-M* ensures that the external entity is permitted access to those assets. Currently, *TF-M* only needs the debug authentication. The whole debug mechanism and related *HAL* will be enhanced in the future. Please refer to the *Debug authentication settings section* for more details.

Design Principles

As *TF-M* runs on resource-constrained devices, the *HAL* tries to avoid multiple level abstractions which cost more resources.

Part of the *HAL* interfaces does not focus on exact hardware operations such as power on/off or PIN manipulation. Instead, the *HAL* abstracts higher-level interfaces to reserve the implementation flexibility for the platform vendors.

The *TF-M HAL* should be easy to deprecate APIs and provide compatibilities. Any API incompatibility should be detected during building.

TF-M relies on the *HAL* APIs to be implemented correctly and trusts the *HAL* APIs. *TFM* can provide assertions to detect common programming errors but essentially no further extensive checks will be provided.

Source Files

This section describes the source file of the *TF-M HAL*, including the header and c files.

tfm_hal_defs.h

This header file contains the definitions of common macros and types used by all *HAL* APIs. Please refer to *Status Codes* for detailed definitions.

tfm_hal_[module].[h/c]

All other headers and c files are classified by the modules, such as isolation, platform, interrupt, devices, etc.

Note: There are common files in the platform folder include the implemented *HAL* APIs. The platform vendors can use them directly but need to implement all the sub APIs.

Status Codes

These are common status and error codes for all *HAL* APIs.

Types

tfm_hal_status_t

This is a status code to be used as the return type of *HAL* APIs.

```
enum tfm_hal_status_t {
    TFM_HAL_ERROR_MEM_FAULT = SCHAR_MIN,
    TFM_HAL_ERROR_MAX_VALUE,
    TFM_HAL_ERROR_BAD_STATE,
    TFM_HAL_ERROR_NOT_SUPPORTED,
    TFM_HAL_ERROR_INVALID_INPUT,
    TFM_HAL_ERROR_NOT_INIT,
    TFM_HAL_ERROR_GENERIC,
    TFM_HAL_SUCCESS = 0
};
```

Error Codes

Negative values indicate an error. Zero and positive values indicate success.

Here is the general list. The detailed usages for each error code are described in the API introduction sections.

TFM_HAL_SUCCESS

Status code to indicate general success.

TFM_HAL_ERROR_GENERIC

Status code to indicate an error that does not correspond to any defined failure cause.

TFM_HAL_ERROR_NOT_INIT

Status code to indicate that the module is not initialed.

TFM_HAL_ERROR_INVALID_INPUT

Status code to indicate that the input is invalid.

TFM_HAL_ERROR_NOT_SUPPORTED

Status code to indicate that the requested operation or a parameter is not supported.

TFM_HAL_ERROR_BAD_STATE

Status code to indicate that the requested action cannot be performed in the current state.

TFM_HAL_ERROR_MAX_VALUE

Status code to indicate that the current number has got the max value.

TFM_HAL_ERROR_MEM_FAULT

Status code to indicate that the memory check failed.

API Definition for TF-M SPM

This section describes the APIs defined for *TF-M SPM*.

Platform API

The platform API is a higher-level abstraction layer of the platform, other than a dedicated API set for the special hardware devices.

APIs

tfm_hal_platform_init()

Prototype

```
enum tfm_hal_status_t tfm_hal_platform_init(void)
```

Description

This API performs the platform initializations **before** the *SPM* initialization.

The initializations could include but not limited to: - Fault handlers - Reset configurations - Debug init - NVIC init

Parameter

- void - None.

Return Values

- TFM_HAL_SUCCESS - Init success.
- TFM_HAL_ERROR_GENERIC - Generic errors.

tfm_hal_system_reset()

Prototype

```
void tfm_hal_system_reset(void)
```

Description

This API performs a system reset.

The platform can uninitialize some resources before reset.

When CONFIG_TFM_HALT_ON_CORE_PANIC is disabled this function is called to reset the system when a fatal error occurs.

Parameter

- void - None

Return Values

- void - None

Note

This API should not return.

tfm_hal_system_halt()

Prototype

```
void tfm_hal_system_halt(void)
```

Description

This API enters the CPU into an infinite loop.

The platform can uninitialize some resources before looping forever.

When CONFIG_TFM_HALT_ON_CORE_PANIC is enabled this function is called to halt the system when a fatal error occurs.

Parameter

- void - None

Return Values

- void - None

Note

This API should not return.

Isolation API

The *PSA-FF-M* defines three isolation levels and a memory access rule to provide diverse levels of security. The isolation API provides the functions to implement these requirements.

The Isolation API operates on boundaries. A boundary represents a set of protection settings that isolates components and domains. Below are the boundary examples in the current implementation:

- Boundaries between SPM and Secure Partitions.
- Boundaries between ARoT domain and PRoT domain.

There are two types of boundaries:

- Static boundaries: Set up when the system is initializing and persistent after the initialization. This type of boundary needs the set-up operations only.
- Partition boundaries: Keeps switching from one to another when the system is running. This type of boundary needs both set-up and switching operations.

The boundary operations are abstracted as HAL interfaces because isolation hardwares can be different for platforms:

- The set-up HAL interface creates a partition boundary based on given partition information. This created boundary is bound with the partition for subsequent usage. The binding is done by storing the boundary into partition runtime data.
- The activation HAL interface activates the partition boundary to secure the execution for the partition to be switched. The target partition's information and boundary are given to the activation HAL to accomplish the operation.

The data representing the partition boundary in runtime is defined with the opaque type uintptr_t:

- It is required that one value represents one boundary. The different values represent different boundaries.
- The value is created by HAL implementation with its own-defined encoding scheme.

The HAL implementation defined encoding scheme can be designed for implementation convenience. For example:

- The implementation scheme can encode attribute flags into integer bits. This could help the activation HAL to extract the protection settings quickly from this encoded value, or even write to hardware registers directly in the most ideal case. The initial TF-M Isolation HAL reference implementation applies this scheme.
- The implementation scheme can reference the addresses of isolation hardware description data. This could help the activation HAL to reference the protection settings directly by pointers.

Multiple Secure Partitions can bind with the same boundary value. This flexibility is useful for specific configurations. Take Isolation Level 2 as an example, assigning PRoT and ARoT domain boundaries to respective partitions can make execution more efficient, because switching two partitions in the same domain does not need to change the activated boundary.

The boundary contains the partition's memory accessibility information, hence memory access check shall be performed based on boundary.

Memory Access Attributes

The memory access attributes are encoded as bit fields, you can logic OR them to have a combination of the atrributes, for example TFM_HAL_ACCESS_UNPRIVILEGED | TFM_HAL_ACCESS_READABLE is unprivileged readable. The data type is *uint32_t*.

TFM_HAL_ACCESS_EXECUTABLE

The memory is executable.

```
#define TFM_HAL_ACCESS_EXECUTABLE (1UL << 0)
```

TFM_HAL_ACCESS_READABLE

The memory is readable.

```
#define TFM_HAL_ACCESS_READABLE (1UL << 1)
```

TFM_HAL_ACCESS_WRITABLE

The memory is writable.

```
#define TFM_HAL_ACCESS_WRITABLE (1UL << 2)
```

TFM_HAL_ACCESS_UNPRIVILEGED

The memory is unprivileged mode accessible.

```
#define TFM_HAL_ACCESS_UNPRIVILEGED (1UL << 3)
```

TFM_HAL_ACCESS_DEVICE

The memory is a MMIO device.

```
#define TFM_HAL_ACCESS_DEVICE (1UL << 4)
```

TFM_HAL_ACCESS_NS

The memory is accessible from *NSPE*

```
#define TFM_HAL_ACCESS_NS (1UL << 5)
```

APIs

tfm_hal_verify_static_boundaries()

Prototype

```
fih_int tfm_hal_verify_static_boundaries(void)
```

Description

This API verifies the static isolation boundaries.

Refer to the *PSA-FF-M* for the definitions of the isolation boundaries.

Parameter

- void - None

Return Values

- TFM_HAL_SUCCESS - Verification has been successful.

- TFM_HAL_ERROR_GENERIC - Verification failed.

tfm_hal_set_up_static_boundaries()

Prototype

```
enum tfm_hal_status_t tfm_hal_set_up_static_boundaries(void)
```

Description

This API sets up the static isolation boundaries which are constant throughout the system runtime.

These boundaries include:

- The boundary between the *SPE* and the *NSPE*
- The boundary to protect the SPM execution. For example, the PSA RoT isolation boundary between the PSA Root of Trust and the Application Root of Trust which is for isolation level 2 and 3 only.

Refer to the *PSA-FF-M* for the definitions of the isolation boundaries.

Parameter

- void - None

Return Values

- TFM_HAL_SUCCESS - Isolation boundaries have been set up.
- TFM_HAL_ERROR_GENERIC - Failed to set up the static boundaries.

tfm_hal_bind_boundary()

Prototype

```
enum tfm_hal_status_t tfm_hal_bind_boundary(
                            const struct partition_load_info_t *p_ldinf,
                            uintptr_t *p_boundary);
```

Description

This API binds partition with a platform-generated boundary. The boundary is bound by writing the generated value into p_boundary. And this bound boundary is used in subsequent calls to *tfm_hal_activate_boundary()* when boundary's owner partition get scheduled for running.

Parameter

- p_ldinf - Load information of the partition that is under loading.
- p_boundary - Pointer for holding a partition's boundary.

Return Values

- TFM_HAL_SUCCESS - The boundary has been bound successfully.
- TFM_HAL_ERROR_GENERIC - Failed to bind the handle.

tfm_hal_activate_boundary()

Prototype

```
enum tfm_hal_status_t tfm_hal_activate_boundary(
                            const struct partition_load_info_t *p_ldinf,
                            uintptr_t boundary);
```

Description

This API requires the platform to activate the boundary to ensure the given Secure Partition can run successfully.

The access permissions outside the boundary is platform-dependent.

Parameter

- p_ldinf - The load information of the partition that is going to be run.
- boundary - The boundary for the owner partition of p_ldinf. This value is bound in function tfm_hal_bind_boundary.

Return Values

- TFM_HAL_SUCCESS - the isolation boundary has been set up.
- TFM_HAL_ERROR_GENERIC - failed to set up the isolation boundary.

tfm_hal_memory_check()

Prototype

```
tfm_hal_status_t tfm_hal_memory_check(uintptr_t boundary,
                                      uintptr_t base,
                                      size_t size,
                                      uint32_t access_type)
```

Description

This API checks if a given range of memory can be accessed with specified access types in boundary. The boundary belongs to a partition which contains asset info.

Parameter

- boundary - Boundary of a Secure Partition. Check *tfm_hal_bind_boundary* for details.
- base - The base address of the region.
- size - The size of the region.
- access_type - The memory access types to be checked between given memory and boundaries. The *Memory Access Attributes*.

Return Values

- TFM_HAL_SUCCESS - The memory region has the access permissions.
- TFM_HAL_ERROR_MEM_FAULT - The memory region does not have the access permissions.
- TFM_HAL_ERROR_INVALID_INPUT - Invalid inputs.
- TFM_HAL_ERROR_GENERIC - An error occurred.

Note

If the implementation chooses to encode a pointer as the boundary, a platform-specific pointer validation needs to be considered before referencing the content in this pointer.

tfm_hal_boundary_need_switch()

Prototype

```
bool tfm_hal_boundary_need_switch(uintptr_t boundary_from,
                                  uintptr_t boundary_to)
```

Description

This API let the platform decide if a boundary switch is needed.

Parameter

- boundary_from - Boundary to switch from.
- boundary_to - Boundary to switch to.

Return Values

- true - A switching is needed
- false - No need for a boundary switch

Log API

The log API is used by the *TF-M log system*. The log device could be uart, memory, usb, etc.

APIs

tfm_hal_output_partition_log()

Prototype

```
int32_t tfm_hal_output_partition_log(const unsigned char *str, uint32_t len)
```

Description

This API is called by Secure Partition to output logs.

Parameter

- str - The string to output.
- len - Length of the string in bytes.

Return Values

- Positive values - Number of bytes output.
- TFM_HAL_ERROR_NOT_INIT - The log device has not been initialized.
- TFM_HAL_ERROR_INVALID_INPUT - Invalid inputs when str is NULL or len is zero.

Note

None.

tfm_hal_output_spm_log()

Prototype

```
int32_t tfm_hal_output_spm_log(const unsigned char *str, uint32_t len)
```

Description

This API is called by *SPM* to output logs.

Parameter

- str - The string to output.
- len - Length of the string in bytes.

Return Values

- Positive numbers - Number of bytes output.
- TFM_HAL_ERROR_NOT_INIT - The log device has not been initialized.

- TFM_HAL_ERROR_INVALID_INPUT - Invalid inputs when str is NULL or len is zero.

Note

Please check *TF-M log system* for more information.

Interrupt APIs

The SPM HAL interrupt APIs are intended for operations on Interrupt Controllers of platforms.

APIs for control registers of interrupt sources are not in the scope of this set of APIs. Secure Partitions should define the APIs for managing interrupts with those MMIO registers.

APIs

tfm_hal_irq_enable()

Prototype

```
enum tfm_hal_status_t tfm_hal_irq_enable(uint32_t irq_num)
```

Description

This API enables an interrupt from the Interrupt Controller of the platform.

Parameter

- irq_num - the interrupt to be enabled with a number

Return Values

- TFM_HAL_ERROR_INVALID_INPUT - the irq_num exceeds The maximum supported number of external interrupts.
- TFM_HAL_ERROR_GENERIC - failed to enable the interrupt.
- TFM_HAL_SUCCESS - the interrupt is successfully enabled.

tfm_hal_irq_disable()

Prototype

```
enum tfm_hal_status_t tfm_hal_irq_disable(uint32_t irq_num)
```

Description

This API disables an interrupt from the Interrupt Controller of the platform.

Parameter

- irq_num - the interrupt to be disabled with a number

Return Values

- TFM_HAL_ERROR_INVALID_INPUT - the irq_num exceeds The maximum supported number of external interrupts.
- TFM_HAL_ERROR_GENERIC - failed to disable the interrupt.
- TFM_HAL_SUCCESS - the interrupt is successfully disabled.

tfm_hal_irq_clear_pending()

Prototype

```
enum tfm_hal_status_t tfm_hal_irq_clear_pending(uint32_t irq_num)
```

Description

This API clears an active and pending interrupt.

Parameter

- irq_num - the interrupt to be disabled with a number

Return Values

- TFM_HAL_ERROR_INVALID_INPUT - the irq_num exceeds The maximum supported number of external interrupts.
- TFM_HAL_ERROR_GENERIC - failed to clear the pending interrupt.
- TFM_HAL_SUCCESS - the pending interrupt has been cleared.

Initializations

Prototype

```
enum tfm_hal_status_t {source_symbol}_init(void *p_pt,
                                           const struct irq_load_info_t *p_ildi)
```

The {source_symbol} is:

- irq_{source}, if the source attribute of the IRQ in Partition manifest is a number
- Lowercase of source attribute, if source is a symbolic name

Description

Each interrupt has an initialization function individually. The *SPM* calls the functions while loading the Partitions.

The following initializations are required for each interrupt:

- Setting the priority. The value must between 0 to 0x80 exclusively.
- Ensuring that the interrupt targets the Secure State.
- Saving the input parameters for future use.

Platforms can have extra initializations as well.

Parameter

- p_pt - pointer to Partition runtime struct of the owner Partition
- p_ildi - pointer to irq_load_info_t struct of the interrupt

Note

Please refer to the :doc: *IRQ intergration guide<tfm_secure_irq_integration_guide>* for more information.

API Definition for Secure Partitions

The Secure Partition (SP) *HAL* mainly focuses on two parts:

- Peripheral devices. The peripherals accessed by the *TF-M* default Secure Partitions.
- Secure Partition abstraction support. The Secure Partition data that must be provided by the platform.

The Secure Partition abstraction support will be introduced in the peripheral API definition.

ITS and PS flash API

There are two kinds of flash:

- Internal flash. Accessed by the PSA Internal Trusted Storage (ITS) service.
- External flash. Accessed by the PSA Protected Storage (PS) service.

The ITS HAL for the internal flash device is defined in the tfm_hal_its.h header and the PS HAL in the tfm_hal_ps.h header.

Macros

Internal Trusted Storage

TFM_HAL_ITS_FLASH_DRIVER

Defines the identifier of the CMSIS Flash ARM_DRIVER_FLASH object to use for ITS. It must have been allocated by the platform and will be declared extern in the HAL header.

TFM_HAL_ITS_PROGRAM_UNIT

Defines the size of the ITS flash device's physical program unit (the smallest unit of data that can be individually programmed to flash). It must be equal to TFM_HAL_ITS_FLASH_DRIVER.GetInfo()->program_unit, but made available at compile time so that filesystem structures can be statically sized.

TFM_HAL_ITS_FLASH_AREA_ADDR

Defines the base address of the dedicated flash area for ITS.

TFM_HAL_ITS_FLASH_AREA_SIZE

Defines the size of the dedicated flash area for ITS in bytes.

TFM_HAL_ITS_SECTORS_PER_BLOCK

Defines the number of contiguous physical flash erase sectors that form a logical filesystem erase block.

Protected Storage

TFM_HAL_PS_FLASH_DRIVER

Defines the identifier of the CMSIS Flash ARM_DRIVER_FLASH object to use for PS. It must have been allocated by the platform and will be declared extern in the HAL header.

TFM_HAL_PS_PROGRAM_UNIT

Defines the size of the PS flash device's physical program unit (the smallest unit of data that can be individually programmed to flash). It must be equal to TFM_HAL_PS_FLASH_DRIVER.GetInfo()->program_unit, but made available at compile time so that filesystem structures can be statically sized.

TFM_HAL_PS_FLASH_AREA_ADDR

Defines the base address of the dedicated flash area for PS.

TFM_HAL_PS_FLASH_AREA_SIZE

Defines the size of the dedicated flash area for PS in bytes.

TFM_HAL_PS_SECTORS_PER_BLOCK

Defines the number of contiguous physical flash erase sectors that form a logical filesystem erase block.

Optional definitions

The TFM_HAL_ITS_FLASH_AREA_ADDR, TFM_HAL_ITS_FLASH_AREA_SIZE and TFM_HAL_ITS_SECTORS_PER_BLOCK definitions are optional. If not defined, the platform must implement tfm_hal_its_fs_info() instead.

Equivalently, tfm_hal_its_ps_info() must be implemented by the platform if TFM_HAL_ITS_FLASH_AREA_ADDR, TFM_HAL_ITS_FLASH_AREA_SIZE or TFM_HAL_ITS_SECTORS_PER_BLOCK is not defined.

Objects

ARM_DRIVER_FLASH

The ITS and PS HAL headers each expose a CMSIS Flash Driver instance.

```
extern ARM_DRIVER_FLASH TFM_HAL_ITS_FLASH_DRIVER

extern ARM_DRIVER_FLASH TFM_HAL_PS_FLASH_DRIVER
```

The CMSIS Flash Driver provides the flash primitives ReadData(), ProgramData() and EraseSector() as well as GetInfo() to access flash device properties such as the sector size.

Types

tfm_hal_its_fs_info_t

Struct containing information required from the platform at runtime to configure the ITS filesystem.

```
struct tfm_hal_its_fs_info_t {
    uint32_t flash_area_addr;
    size_t flash_area_size;
    uint8_t sectors_per_block;
};
```

Each attribute is described below:

- flash_area_addr - Location of the block of flash to use for ITS
- flash_area_size - Number of bytes of flash to use for ITS
- sectors_per_block - Number of erase sectors per logical FS block

tfm_hal_ps_fs_info_t

Struct containing information required from the platform at runtime to configure the PS filesystem.

```
struct tfm_hal_ps_fs_info_t {
    uint32_t flash_area_addr;
    size_t flash_area_size;
    uint8_t sectors_per_block;
};
```

Each attribute is described below:

- flash_area_addr - Location of the block of flash to use for PS
- flash_area_size - Number of bytes of flash to use for PS

- sectors_per_block - Number of erase sectors per logical FS block

Functions

tfm_hal_its_fs_info()

Prototype

```
enum tfm_hal_status_t tfm_hal_its_fs_info(struct tfm_hal_its_fs_info_t *fs_info);
```

Description

Retrieves the filesystem configuration parameters for ITS.

Parameter

- fs_info - Filesystem config information

Return values

- TFM_HAL_SUCCESS - The operation completed successfully
- TFM_HAL_ERROR_INVALID_INPUT - Invalid parameter

Note

This function should ensure that the values returned do not result in a security compromise. The block of flash supplied must meet the security requirements of Internal Trusted Storage.

tfm_hal_ps_fs_info()

Prototype

```
enum tfm_hal_status_t tfm_hal_ps_fs_info(struct tfm_hal_ps_fs_info_t *fs_info);
```

Description

Retrieves the filesystem configuration parameters for PS.

Parameter

- fs_info - Filesystem config information

Return values

- TFM_HAL_SUCCESS - The operation completed successfully
- TFM_HAL_ERROR_INVALID_INPUT - Invalid parameter

Note

This function should ensure that the values returned do not result in a security compromise.

Copyright (c) 2020-2024, Arm Limited. All rights reserved. Copyright (c) 2022 Cypress Semiconductor Corporation (an Infineon company) or an affiliate of Cypress Semiconductor Corporation. All rights reserved.

Cooperative Scheduling Rules

Author
Ashutosh Singh

Organization
Arm Limited

Contact
Ashutosh Singh <ashutosh.singh@arm.com>

TF-M Scheduler - Rules

On ArmV8-M CPUs, NSPE and SPE share the same physical processing element(PE). A TF-M enabled systems need to be able to handle asynchronous events (interrupts) regardless of current security state of the PE, and that may lead to scheduling decisions. This introduces significant complexity into TF-M. To keep the integrity of (NSPE and SPE) schedulers and call paths between NSPE and SPE, following set of rules are imposed on the TF-M scheduler design.

Objectives and Requirements

1. Decoupling of scheduling decisions between NSPE and SPE
2. Efficient interrupt handling by SPE as well as NSPE
3. Reduce critical sections on the secure side to not block interrupts unnecessarily
4. Scalability to allow simplification/reduction of overheads to scale down the design for constrained devices
5. Reduce impact on NSPE software design

a. NSPE interrupt handling implementation should be independent

b. Impact on the NSPE scheduler should be minimal

c. No assumptions should be made about NSPE's scheduling capabilities

Scheduler Rules for context switching between SPE and NSPE

To allow coherent cooperative scheduling, following set of rules are imposed on security state changes. The switching between SPE and NSPE can be triggered in multiple ways.

Involuntary security state switch; when the software has no control over the switch:

- A NSPE interrupt takes control into NSPE from SPE
- A SPE interrupt takes control into SPE from NSPE

Voluntary security state switch; when software programmatically makes the switch:

- A NSPE exception handler returns from NSPE to pre-empted SPE context
- A SPE exception handler returns from SPE to pre-empted NSPE context
- NSPE makes a function call into SPE
- SPE returns a call from NSPE
- SPE makes a function call into NSPE (not covered in current design)
- NSPE returns a call from SPE (not covered in current design)

In order to maintain the call stack integrity across NSPE and SPE, following rules are imposed on all security state switches.

Rules for NSPE Exception handling

1. **The NSPE exception handler is allowed to trigger a NSPE context switch (regardless of security state of the preempted context.**

This is expected behaviour of any (RT)OS.

2. **The NSPE scheduler must eventually 'restore' the preempted (by exception) context.**

This is expected behaviour of any (RT)OS.

3. **If NSPE exception results in a NSPE context switch, SPM must be informed of the scheduling decision; this must be done BEFORE the execution of newly scheduled-in context.**

This is to ensures integrity of the call stack when SPE is ready to return a previous function call from NSPE.

Rules for SPE Exception handling

1. **All of the SPE interrupts must have higher priority than NSPE interrupts**

This is rule is primarily for simplifying the SPM design.

2. **The SPE interrupt handler is allowed to trigger a SPE context switch (regardless of security state of the pre-empted context)**

If the SPE context targeted by the interrupt is not same as current SPE context, the SPM may choose to switch the current running SPE context based on priority.

3. **SPE scheduler must treat pre-empted context as one of the SPE contexts**

   a. If the pre-empted SPE context is SP1, the TCB for SP1 should be used for saving the context. i.e. the context of SP1 should be saved before scheduling anything other secure partition.

   b. If SP1 was pre-empted by a NSPE interrupt, and subsequent NSPE execution is pre-empted by SPE exception (before NSPE scheduling decision is communicated back to SPM) – SP1 TCB must be used for saving the context In this case SPM is not yet aware of the NSPE context switch, from SPM's standpoint SP1 is still executing, so SPM assumes that the preempted context is SP1.

   c. If SP1 was pre-empted by a NSPE interrupt, and subsequent NSPE execution is pre-empted by SPE exception *after* NSPE scheduling decision is communicated back to SPM) - a TCB dedicated to NSPE should be used for saving the context.

   When NSPE scheduler communicates the scheduling decision to SPM, SPM must save the SP1 context, if a SPE interrupt preempts the currently running NSPE context, SPM should save the context to a dedicated NSPE TCB.

   d. The SPE scheduler must eventually 'restore' the pre-empted context. This is an expected behaviour of any scheduler.

4. **All of the interrupts belonging to a partition must have same priority.**

This serializes ISR execution targeted for same partition.

5. **In case of nested interrupts, all of the ISRs must run to finish before any service code is allowed to run**

This is an expected behaviour of any scheduler.

6. **If the previously preempted context was a NSPE ISR context, SPE ISR handler must return to preempted NSPE context.**

This is an expected behaviour of any scheduler to return to preempted context.

Rules for NSPE to SPM function call (and NSPE scheduler)

1. Current NSPE context must have been communicated to SPM, otherwise SPM cannot guarantee NSPE function calling stack integrity.

Rules for Function Return from SPE to NSPE with result

1. **The result available on SPE side are for currently active NSPE context.**

To maintain call stack integrity, if SPE is ready to return to NSPE, it can do function return only if the SPE return path corresponds to currently active NSPE context.

2. **Last entry into secure world happened programmatically (Voluntary security state switch into SPE)**

i.e. control is voluntarily given back by NSPE, either through a function call, or a context restore via 'return to SPE from NSPE'. As opposed to a SPE interrupt bringing back the execution into SPE.

3. **The current NSPE call stack has not already been returned with SPM_IDLE.**

This rule applies if following optional feature is enabled.

Rules for Return from SPE to NSPE with SPM_IDLE

This is optional part of the design as it introduces significant complexity on both sides of the security boundary. It allows yielding of the CPU to NSPE when SPE has not CPU execution to do but it has not yet finished the previous request(s) from NSPE; i.e. SPE is waiting on arrival of a SPE interrupt.

1. **Last entry into secure world happens programmatically (Voluntary security context switch into SPE)**

i.e. control is voluntarily given back by NSPE, either through a function call, or a context restore via 'return to SPE from NSPE'. As opposed to a SPE interrupt bringing back the execution into SPE.

2. **The result for the currently active NSPE entity is not yet available, the called service is waiting (on interrupt/event).**

SPE request corresponding to currently active NSPE caller is not yet completed and is waiting on an ISR.

3. **The current NSPE call stack has not already been returned with SPM_IDLE.**

Rules for NSPE pend irq based return from SPE to NSPE

This is optional part of the design as it introduces significant complexity on both sides. This works in conjunction with [Rules for Return from SPE to NSPE with SPM_IDLE](#rules-for-return-from-spe-to-nspe-with-spm_idle). In this scenario, when SPE is ready with result for a previous call from NSPE, it raises a pended IRQ to NSPE instead of returning the function call path.

1. **The SPE has finished a NSPE request.**
2. **The corresponding NSPE context has already been returned with SPM_IDLE.**

Rules for ISR pre-emption

1. **A higher priority NSPE interrupt is allowed to preempt a lower priority NSPE ISR**
2. **A higher priority SPE interrupt is allowed to preempt a lower priority SPE ISR**
3. **A SPE interrupt is allowed to preempt NSPE ISR**
4. **A NSPE interrupt is not allowed to preempt SPE ISR**
5. **All interrupts belonging to a service must have same priority**

Copyright (c) 2019-2024, Arm Limited. All rights reserved.

Code Generation With Jinja2

Author
Mate Toth-Pal

Organization
Arm Limited

Contact
Mate Toth-Pal <mate.toth-pal@arm.com>

Generating files from templates in TF-M

Some of the files in TF-M are generated from template files. The files to be generated are listed in tools/tfm_generated_file_list.yaml. For each generated file <path_to_file>/<filename> the template file is <path_to_file>/<filename>.template. The templates are populated with partition information from the partition manifests. The manifests to be used for the generation are listed in tools/tfm_manifest_list.yaml.

Custom generator script - Current method

tools/tfm_parse_manifest_list.py Python script is used to generate files from the templates. This script calls the tools/generate_from_template.py to parse the template files, and uses tools/keyword_substitution.py to substitute the keychains with actual values from the manifest files.

Use Jinja2 template engine - proposal

The proposal is to eliminate the template parser and substituter scripts, and call the Jinja2 template engine library from tools/tfm_parse_manifest_list.py to do the substitution.

More information on jinja2: http://jinja.pocoo.org/

Changes needed:

- tools/tfm_parse_manifest_list.py have to be modified to call the Jinja2 library instead of the custom scripts. The data structure required by the library is very similar to the one required by the current scripts.
- template files needs to be rewritten to the Jinja syntax: The control characters to be replaced (like @! -> {%) and for statements to be added to iterate over the substitutions.

Improvements over the current solution

- Compatible with Python 2.7 and Python 3, while current solution only supports Python 2.7
- More advanced functionality: direct control over the list of items for a keychain, meta information on that list (like length)
- Well documented (see website)
- Jinja2 is free and open-source software, BSD licensed, just like TF-M. It is a well established software in contrast with the current proprietary solution.

Example

Below code snippet enumerates services in Secure Partitions:

```
{% for partition in partitions %}
   {% if partition.manifest.services %}
      {% for service in partition.manifest.services %}
         {do something for the service}
      {% endfor %}
   {% endif %}
{% endfor %}
```

Copyright (c) 2019-2021, Arm Limited. All rights reserved.

Fixing implicit casting for C enumeration values

Author
Hugues de Valon

Organization
Arm Limited

Contact
hugues.devalon@arm.com

Abstract

C enumerations provide a nice way to increase readability by creating new enumerated types but the developer should take extra care when mixing enumeration and integer values. This document investigates C enumerations safety and proposes strategies on how to fix the implicit casting of the enumeration values of TF-M with other types.

C99 Standard point of view

In TF-M many implicit casts are done between integer types (uint32_t, int32_t, size_t, etc), enumerated types (enum foobar) and enumeration constants (FOOBAR_ENUM_1).

According to the C99 standard[1]:

> **§6.2.5, 16**: An enumeration comprises a set of named integer constant values. Each distinct enumeration constitutes a different numerated type.
>
> **§6.7.2.2, 2**: The expression that defines the value of an enumeration constant shall be an integer constant expression that has a value representable as an int.
>
> **§6.7.2.2, 3**: The identifiers in an enumerator list are declared as constants that have type int and may appear wherever such are permitted.
>
> **§6.7.2.2, 4**: Each enumerated type shall be compatible with char, a signed integer type, or an unsigned integer type. The choice of type is implementation-defined, but shall be capable of representing the values of all the members of the enumeration.

From these four quotes from the C99 standard[Page 1468, 1], the following conclusions can be made:

- an enumeration defines a new type and should be treated as such
- the enumeration constants must only contains value representable as an int
- the enumeration constants have type int
- the actual type of the enumeration can be between char, signed and unsigned int. The compiler chooses the type it wants among those that can represent all declared constants of the enumeration.

Example:

```
enum french_cities {
    MARSEILLE,
    PARIS,
    LILLE,
    LYON
};
```

In that example, MARSEILLE, PARIS, LILLE and LYON are enumeration constants of type int and enum french_cities is a enumerated type which can be of actual type char, unsigned int or int (the compiler chooses!).

For these reasons, doing an implicit cast between an enumeration and another type is the same as doing an implicit cast between two different types. From a defensive programming point of view, it should be checked that the destination type can represent the values from the origin type. In this specific case it means four things for enumerations:

- it is always safe to assign an enumeration constant to an int, but might be better to cast to show intent.

[1] C99 standard: http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf

- when casting an enumeration constant to another type, it should be checked that the constant can fit into the destination type.
- when casting from an integer type (uint32_t, int32_t, etc) to an enumeration type, it should be checked that the integer's value is one of the enumeration constants. The comparison needs to be done on the biggest type of the two so that no information is lost. C integer promotion should automatically do that for the programmer (check §6.3.1.8, 1 for the rules).
- when casting from an enumeration type to an integer type, it should be checked that the enumeration type value fits into the integer type. The value of a variable which has the type of an enumeration type is not limited to the enumeration constants of the type. An enumeration constant will always fit into an int.

Strategies to fix

0. Replace the enumerated type with an integer type and replace the enumeration constant with preprocessor constants.
1. Whenever possible, try to use matching types to avoid implicit casting. It happens, for example, for arithmetic operations, function calls and function returns. This strategy always have the lowest performance impact.
2. When using an enumeration constant in an arithmetic operation with another type, verify that the constant can fit in the other type and cast it.
3. When converting an integer to an enumeration type, use a conversion function to check if the integer matches an enumeration constant. To not impact performance too much, this function should be an inline function. If it does not match, use (or add) the error constant or return an error value.
4. When converting an enumeration type to an integer, use a conversion function to check that the integer type can contain the enumeration value.

Design proposal for TF-M

In TF-M, an action will be taken for all enumerated types and enumeration constants that are used for implicit casting. The goal of this proposal is to remove all implicit casting of enumeration values in TF-M.

The following enumerated types will be removed and replaced with preprocessor constants (strategy 0). These enumerated types are not used in TF-M but only the constants they declare.

- enum spm_part_state_t
- enum spm_part_flag_mask_t
- enum tfm_partition_priority

The following enumerated types will be kept because they are used in the prototypes of functions and are useful for debugging. Whenever possible, strategy 1 will be applied to remove implicit casting related with those enumerations but dynamic conversions will be used if the first option would create too much change in the code base.

- enum tfm_status_e: the return type of the following functions will be changed to return the enum tfm_status_e type. These functions are already returning the enumeration constants, but implicitly casted to an integer type like int32_t.
  - int32_t check_address_range
  - int32_t has_access_to_region
  - int32_t tfm_core_check_sfn_parameters
  - int32_t tfm_start_partition
  - int32_t tfm_return_from_partition

  - int32_t tfm_check_sfn_req_integrity
  - int32_t tfm_core_check_sfn_req_rules
  - int32_t tfm_spm_sfn_request_handler
  - int32_t tfm_spm_sfn_request_thread_mode
- enum tfm_buffer_share_region_e: the following function prototypes will be changed:
  - tfm_spm_partition_set_share(uint32_t partition_idx, uint32_t share) -> tfm_spm_partition_set_share(uint32_t partition_idx, enum tfm_buffer_share_region_e share)
- enum tfm_memory_access_e
- enum attest_memory_access_t
- enum engine_cipher_mode_t
- mbedtls_cipher_type_t

The following enumerated types are used for error code values of Secure service calls. They should be kept as they are part of the interface and might be used by external parties in Non-Secure code. For the Initial Attestation service, the enumeration is defined in the PSA Attestation API specifications.

- enum psa_attest_err_t
- enum psa_audit_err
- enum tfm_platform_err_t

Implicit casting related with these enumerations is happening in two locations of TF-M and need conversion functions in those locations, because the types can not be changed:

- In the Non-Secure Client library, all of the Secure Service functions implicitly cast the uint32_t returned by tfm_ns_lock_dispatch to these enumerated types. Strategy 3 is needed here.
- In all of the veneer functions, there is an implicit cast from the int32_t value returned by the SVC request function (tfm_core_*_request) to these enumerated types. Strategy 3 is needed here as well. The implicit cast will eventually be removed if all of the services are using the Uniform Signatures Prototypes so that the veneer functions all return psa_status_t which is an int32_t.

If the interface of those services can be changed, these enumerations could be removed and replaced with the psa_status_t type to remove the implicit casting.

Copyright (c) 2019-2020, Arm Limited. All rights reserved.

Copyright (c) 2023-2024, Arm Limited. All rights reserved.

FF-M Isolation

Organization
Arm Limited

Contact
tf-m@lists.trustedfirmware.org

This document analyzes the isolation rules of implementing FF-M 1.0 isolation and introduces the reference implementation in TF-M, which complies the rules by operating the hardware and software resources.

Note: Reference the document *Glossary* for terms and abbreviations.

Introduction

This chapter describes the definitions from FF-M and analyzes the possible implementation key-points.

Isolation Levels

There are 3 isolation levels (1-3) defined in FF-M, the greater level number has more isolation boundaries.

The definition for Isolation Level 1:

- L1.1 NPSE needs protection from nobody.
- L1.2 SPE needs protection from NSPE.

The definition for Isolation Level 2:

- L2.1 NPSE needs protection from nobody.
- L2.2 Application Root of Trust (ARoT) needs protection from NSPE.
- L2.3 PSA Root of Trust (PRoT) needs protection from NSPE and ARoT.

The definition for Isolation Level 3:

- L3.1 NPSE needs protection from nobody.
- L3.2 Secure Partition needs protection from NSPE and other Secure Partitions.
- L3.3 PSA Root of Trust (RoT) domain needs protection from NSPE and all Secure Partitions.

Important: A Secure Partition RoT Service is a Root of Trust Service implemented within a Secure Partition. An Application RoT Service must be implemented as a Secure Partition RoT Service. But it is implementation-defined whether a PSA RoT Service is a Secure Partition RoT Service.

Here listed several possible PSA RoT Service implementation mechanisms:

1. Implement PSA RoT Services in Secure Partitions with respective boundaries.
2. Implement PSA RoT Services in Secure Partitions, but no boundaries between these Secure Partitions as they are in the PSA RoT Domain.
3. Implement PSA RoT Services in a customized way instead of Secure Partitions, an internal library of PSA RoT domain e.g.

TF-M chooses the 2nd option to balance performance and complexity.

Isolation Rules

The essence of isolation is to protect the assets of one protection domain from being accessed from other domains. The isolation levels define where the isolation boundaries should be placed, the isolation rules define the strength of the isolation the boundaries should offer.

Note: Refer to chapter *Memory Assets* in Firmware Framework for M (FF-M) to know asset class items. Assets are represented by memory addresses in the system memory map, which makes assets named *Memory Assets*. The often-seen asset items are ROM, RAM, and memory-mapped peripherals.

Memory Asset Class

There are 3 memory asset classes defined in Firmware Framework for M (FF-M):

- Code
- Constant data
- Private data

There are 6 isolation rules for protecting assets. Here lists the simplified description for the rules, check chapter 3.1.2 of FF-M 1.0 for the original description:

- I1. Only Code is executable.
- I2. Only private data is writable.
- I3. If domain A needs protection from domain B, then Private data in domain A cannot be accessed by domain B.
- I4. (Optional) If domain A needs protection from domain B, then Code and Constant data in domain A is not readable or executable by domain B.
- I5. (Optional) Code in a domain is not executable by any other domain.
- I6. (Optional) All assets in a domain are private to that domain and cannot be accessed by any other domain, with the following exception: The domain containing the SPM can only access Private data and Constant data assets of other domains when required to implement the PSA Firmware Framework API.
- I7. (Optional, added in FF-M 1.1) Constant data is not executable.

  The first 3 rules from I1 to I3 defines the mandatory rules to comply the isolation, while I4 to I6 are optional rules to enhance the isolation boundaries.

Important: There is a table in the chapter 3.1.2 of FF-M 1.0 under I1 lists the asset types and allowed access method. Preventing executable access on constant data costs more hardware resources, so there is an optional rule I7 created in FF-M Extensions (FF-M 1.1) to comfort implementations with constrained hardware resources.

Hardware Infrastructure

To implement a secure system, the hardware security framework (TrustZone or multiple-core e.g.) and their auxiliary components (SAU e.g.) are required to ensure the isolation between SPE and NSPE. The requirements:

Important: The interface between secure and non-secure states needs to be fully enumerated and audited to prove the integrity of the secure state isolation.

Besides this SPE and NSPE isolation mechanism, the following analyzes the implementation rules to find out the hardware requirements for isolation inside SPE domains:

- I1 and I2: The assets can be categorized into 3 *Memory Asset Class*, each type has the specific access rules.
- I3: The private data access from the prevented domain needs to be blocked.
- I4: All the assets access from the prevented domain needs to be blocked.
- I5: Code execution from all other domains (even the domain not prevented from) needs to be blocked.
- I6: All the assets access from all other domains (includes non-prevented domain) needs to be blocked, but, SPM is an exception, which can access the private data and constant data of the current domain.

The above items list the requirements for memory access, here are two more points:

- If the memory device or the peripheral are shared between multiple hosts (Such as multiple CPU or DMA, etc), specific hardware protection units need to be available for validating accesses to that device or peripheral.
- The MMIO range for Secure Partitions is not allowed to be overlapped, which means each partition should have exclusive memory-mapped region if they require a peripheral device. The memory-mapped region is regarded as the private data so access to this area needs to be validated.

Reference Implementation

This chapter describes the isolation implementation inside SPE by using the Armv8m architecture component - Memory Protection Unit (MPU). The MPU can isolate CPU execution and data access.

Note: Previous version M-profile architecture MPU setting is similar in concept but the difference in practical register formats, which is not described in this document.

The MPU protects memory assets by regions. Each region represents a memory range with specific access attributes.

Note: The maximum numbers of MPU regions are platform-specific.

The SPM is running under the privileged mode for handling access from services. The MPU region for SPM needs to be available all the time since SPM controls the MPU setting while scheduling.

Since partitions are scheduled by SPM, the MPU regions corresponding to the partitions can be configured dynamically while scheduling. Since there is only one running at a time and all others are deactivated, the SPM needs to set up necessary regions for each asset type in one partition only.

There is re-usable code like the C-Runtime and RoT Service API which are same across different partitions. TF-M creates a Secure Partition Runtime Library (SPRTL) as a specific library shared by the Secure Partition. Please refer to *Secure Partition Runtime Library* for more detail.

Note: Enable SPRTL makes it hard to comply with the rules I4, I5 and I6, duplicating the library code can be one solution but it is not “shared” library anymore.

As mentioned in the last chapter, MMIO needs extra MPU regions as private data.

MPU Region Access Permission

The following content would mention the memory access permission to represent the corresponded asset classes.

These access permissions are available on Armv8m MPU:

- Privileged Read-Only (RO)
- All RO
- Privileged Read-Write (RW)
- All RW
- Execution Never (XN)

And one more Armv8.1M access permssion:

- Privileged Execution Never (PXN)

The available regions type list:

Type	Attributes	Privilege Level	Asset
P_RO	RO	Privileged	PRoT Code
P_ROXN	RO + XN	Privileged	PRoT Constant Data
P_RWXN	RW + XN	Privileged	PRoT Private Data/Peripheral
A_RO	RO	Any privilege	Partition/SPRTL Code
A_ROXN	RO + XN	Any privilege	Partition/SPRTL Constant Data
A_RWXN	RW + XN	Any privilege	Partition/SPRTL Private Data/Peripheral
A_ROPXN	RO + PXN	Any privilege	Armv8.1M Partition/SPRTL Code

Example Image Layout

The secure firmware image contains components such as partitions, SPM and the shared code and data. Each component may have different class assets. There would be advantages if placing the assets from all components with the same access attributes into one same region:

- The data relocating or clearing when booting can be done in one step instead of breaking into fragments.
- Assets with statically assigned access attribute can share the same MPU region which saves regions.

Take the TF-M existing implementation image layout as an example:

```
 Level 1      Level 2             Level 3
 Boundaries   Boundaries          Boundaries
+-----------+---------+-----------------------------------+
|           |         |   PRoT SPM                 Code   |
|           | PRoT    +-----------------------------------+
|           | Code    |   PRoT Service             Code   |
|  Code     +---------+-----------------------------------+
|  (ROM)    |         |   Partition 1              Code   |
|           |         +-----------------------------------+
|           | ARoT    |   Partition N              Code   |
|           | Code    +-----------------------------------+
|           |         |   SPRTL                    Code   |
+-----------+---------+-----------------------------------+
Check [4] for more details between Code and Constant Data.
+-----------+---------+-----------------------------------+
|           | PRoT    |   PRoT SPM        Constant Data   |
|           | Constant+-----------------------------------+
|           | Data    |   PRoT Service    Constant Data   |
|  Constant +---------+-----------------------------------+
|   Data    | ARoT    |   Partition 1     Constant Data   |
|   (ROM)   | Constant+-----------------------------------+
|           | Data    |   Partition N     Constant Data   |
|           |         +-----------------------------------+
|           |         |   SPRTL           Constant Data   |
+-----------+---------+-----------------------------------+

+-----------+---------+-----------------------------------+
|           | PRoT    |   PRoT SPM        Private Data    |
|           | Private +-----------------------------------+
|           | Data    |   PRoT Service    Private Data    |
|  Private  +---------+-----------------------------------+
|   Data    |         |   Partition 1     Private Data    |
|   (RAM)   | ARoT    +-----------------------------------+
|           | Private |   Partition N     Private Data    |
```

(continues on next page)

(continued from previous page)

```
|           | Data     +-----------------------------------+
|           |          |  SPRTL          Private Data      |
+-----------+----------+-----------------------------------+
```

Note:

1. Multiple binaries image implementation could also reference this layout if its hardware protection unit can cover the exact boundaries mentioned above.
2. Private data includes both initialized and non-initialized (ZI) sections. Check chapter 3.1.1 of FF-M for the details.
3. This diagram shows the boundaries but not orders. The order of regions inside one upper region can be adjusted freely.
4. As described in the important of *Memory Asset Class*, the setting between Code and Constant Data can be skipped if the executable access method is not applied to constant data. In this case, the groups of Code and Constant Data can be combined or even mixed – but the boundary between PRoT and ARoT are still required under level higher than 1.

Example Region Numbers under Isolation Level 3

The following table lists the required regions while complying the rules for implementing isolation level 3. The level 1 and level 2 can be exported by simplifying the items in level 3 table.

Important: The table described below is trying to be shared between all supported platforms in Trusted Firmware - M. It is obvious that some platforms have special characteristics. In that case, the specific layout table for a particular platform can be totally redesigned but need to fulfil the isolation level requirements.

- Care only the running partitions assets since deactivated partition does not need regions.
- *X* indicates the existence of this region can't comply with the rule.
- An *ATTR* + *n* represent extra n regions are necessary.
- Here assumes the rules with a greater number covers the requirements in the rules with less number.

Here lists the required regions while complying with the rules:

Region Purpose	I1 I2 I3	I4	I5	I6
PRoT SPM Code	A_RO	P_RO	P_RO	P_RO
PRoT Service Code				A_ROPXN
Active Partition Code		A_RO	A_ROPXN	
SPRTL Code		X	X	X
PRoT SPM RO	A_ROXN	P_ROXN	P_ROXN	P_ROXN
PRoT Service RO				A_ROXN
Active Partition RO		A_ROXN	A_ROXN	
SPRTL RO		X	X	X
PRoT SPM RW	P_RWXN	P_RWXN	P_RWXN	P_RWXN
PRoT Service RW				A_RWXN
Active Partition RW	A_RWXN	A_RWXN	A_RWXN	
SPRTL RW [5]	A_RWXN + 1	X	X	X
Partition Peri	A_RWXN + n	A_RWXN + n	A_RWXN + n	A_RWXN + n
Total Numbers	[1]	[2]	[3]	[4]

Note:

1. Total number = A_RO + A_ROXN + P_RWXN + A_RWXN + (1 + n)A_RWXN, while n equals the maximum peripheral numbers needed by one partition. This is the configuration chosen by the reference implementation.
2. Total number = P_RO + P_ROXN + P_RWXN + A_RO + A_ROXN + (1 + n)A_RWXN, the minimal result is *6*, and SPRTL can not be applied.
3. Total number = P_RO + P_ROXN + P_RWXN + A_ROXN + (1 + n)A_RWXN + A_ROPXN, the minimal result is *6*, SPRTL can not be applied, and PXN is required.
4. Total number = P_RO + P_ROXN + P_RWXN + A_ROXN + (1 + n)A_RWXN + A_ROPXN, the minimal result is *6*, SPRTL can not be applied, and PXN is required. To comply with this rule, the PSA RoT Service needs to be implemented as Secure Partitions.
5. This data belongs to SPRTL RW but it is set as Read-Only and only SPM can update this region with the activate partition's metadata for implementing functions with owner SP's context, such as heap functions. This region can be skipped if there is no metadata required (such as no heap functionalities required).

The memory-mapped regions for peripherals have different memory access attributes in general, they are standalone regions in MPU even their attributes covers 'A_RWXN'.

Default access rules

Hardware protection components MAY have the capability to collect regions not explicitly configured in static or runtime settings, and then apply default access rules to the collected. Furthermore, one default rule can be applied to multiple non-contiguous regions which makes them share a common boundary. This operation sets up a standalone 'region' as same as other explicitly configured regions. And it doesn't affect the analysis summary above - just be aware that some regions listed in the table MAY not be explicitly configured.

Take the MPU as an example, MPU can assign a default privileged access attribute to the regions (SPM and PRoT regions e.g.) not explicitly configured. This feature can reduce required MPU regions and ease the programming because regions can be put non-address-contiguous and skip the explicit configuration.

Important: When this default access rules mechanism is applied, the non-explicitly-expressed regions must be reviewed to ensure the isolation boundaries are set properly.

Interfaces

The isolation implementation is based on the HAL framework. The SPM relies on the HAL API to perform the necessary isolation related operations.

The requirement the software need to do are these:

- Create enough isolation protection at the early stage of system booting, just need to focus on the SPM domain.
- Create an isolation domain between secure and non-secure before the jump to the non-secure world.
- Create an isolation domain for each Secure Partition after the Secure Partition is loaded and before jumping to its entry point. The isolation domain should cover all the assets of the Secure Partition, include all its memory, interrupts, and peripherals.
- Switch isolation domains when scheduling different Secure Partitions.

- It is also a requirement that the platform needs to help to check if the caller of the PSA APIs is permitted to access some memory ranges.

The design document *TF-M Hardware Abstraction Layer* gives a detail design, include the platform initialization, isolation interfaces. Please refer to it for more detail.

Appendix

Firmware Framework for M (FF-M)

FF-M Extensions (FF-M 1.1)

Trusted Base System Architecture for M (TBSA-M)

Copyright (c) 2020-2022, Arm Limited. All rights reserved.

TF-M builtin keys

Author
Raef Coles

Organization
Arm Limited

Contact
raef.coles@arm.com

Introduction

When TF-M is integrated on a platform, the platform itself can provide several keys for usage which are bound to the device instead of being owned by a specific secure partition. When those keys are readable by the secure processing environment, the platform must provide a function to load these keys in the HAL layer, either loading them from OTP or another platform specific location or subsystem. These keys are henceforth referred to as "builtin keys", and might include (but are not limited to) the following:

1. The Hardware Unique Key (HUK)
2. The Initial Attestation Key (IAK)

The tfm_builtin_key_loader component implements a mechanism to discover those keys and make them available to the PSA Crypto APIs for usage. Note that if a key is stored in a subsystem which can't be read by the secure processing environment, a full opaque driver[1] must be used to be able to use those keys through the PSA Crypto APIs. This document focuses only on the case where the keys can be read by the SPE (i.e. running TF-M), so the driver applies only on those cases, which can be considered as "_transparent_ builin keys".

In TF-M's legacy solution, the IAK is loaded by the attestation service as a volatile key, which requires some key-loading logic to be implemented by that partition. The HUK is not loaded in the crypto service, and is instead used by an implementation of a TF-M specific KDF algorithm which then loads the key and invokes Mbed TLS directly. Both solutions are far from ideal as they require an effort to load (and duplicate) the keys per each user, or require a dedicated code in TF-M that directly interacts with the HAL layer and invokes functions from the crypto library

[1] PSA cryptoprocessor driver interface: https://github.com/Mbed-TLS/mbedtls/blob/development/docs/proposed/psa-driver-interface.md

bypassing the PSA Crypto interface. The aim of the tfm_builtin_key_loader driver is to provide a uniform interface to use the builtin keys available in a platform.

Implementing a driver to deal with builtin keys allows to expand the legacy solution for dealing with this type of keys in several ways. For example, it avoids the need for partitions to implement dedicated mechanisms for probing the HAL layer for builtin keys and load them as volatile. It removes the need to have implementation specific call flows for deriving keys from the HUK (as that requirement now is fulfilled by the driver itself transparently). It allows uniform access to the keys also from the NS world (in any case, subject to the policy dictated by the platform). Correctly abstracts away details of the key handling mechanism from TF-M partitions into the PSA Crypto core key management subsystem.

PSA builtin keys

The PSA Cryptographic API provides a mechanism for accessing keys that are stored in platform-specific locations (often hardware accelerators or OTP). A builtin key is assigned a specific key_id, i.e. a handle, which is hardcoded at build time and must be selected in the range [MBEDTLS_PSA_KEY_ID_BUILTIN_MIN, MBEDLTS_PSA_KEY_ID_BUILTIN_MAX]. A user of the PSA Crypto API can reference those keys directly by using these handles. It is up to the platform to specify policy restrictions for specific users of the keys, e.g. an NS entity, or a secure partition. The PSA Crypto core will then check those policies to grant or deny access to the builtin key for that user.

PSA cryptoprocessor driver API

The PSA specification allows the PSA Crypto APIs to defer their operation to an accelerator driver, through a mechanism described in the PSA cryptoprocessor driver interface specification[Page 1477, 1]. This specification defines the concept of PSA Crypto core, i.e. “An implementation of the PSA Cryptography API is composed of a core and zero or more drivers”. The PSA specification also has the concept of storage locations for keys, through the type psa_key_location_t, which is used to access keys that don’t have local storage[2]. This is leveraged mainly by opaque drivers mainly that use keys for which the key material is not readable by the PSA crypto core layer.

TF-M defines a software driver called tfm_builtin_key_loader that provides no acceleration but just defines a dedicated key location defined through the TFM_BUILTIN_KEY_LOADER_KEY_LOCATION define. By resorting to the entry points provided by this driver, the PSA Crypto core slot management subsystem can access keys stored in the underlying platform, validate key usage policies and allow PSA Crypto APIs to uniformly access builtin keys using the same call flows used with traditional local-storage keys.

This is implemented by hooking the entry points defined by the driver into the library/psa_crypto_driver_wrappers.c file provided by the PSA Crypto core. This is currently done manually but eventually could be just autogenerated by parsing a description of the driver entry points in the JSON format. These entry points are:

1. tfm_builtin_key_loader_init
2. tfm_builtin_key_loader_get_key_buffer_size
3. tfm_builtin_key_loader_get_builtin_key

The call flow for the entry points from the driver wrapper layer is as follows:

1. During the driver initialisation phase started by the PSA Crypto Core init, the tfm_builtin_key_loader_init function is called to probe the platform HAL and retrieve the builtin keys. Those keys are described by the types defined in the HAL layer interface header tfm_plat_crypto_keys.h. In particular global tables describing key properties and user policies must be implemented by each platform and are retrieved by the two accessor functions tfm_plat_builtin_key_get_desc_table_ptr() and

[2] Definition of psa_key_location_t type in the PSA spec: https://arm-software.github.io/psa-api/crypto/1.1/api/keys/lifetimes.html#c.psa_key_location_t

tfm_plat_builtin_key_get_policy_table_ptr(). The keys are loaded from the platform in secure RAM in the TF-M Crypto partition with associated metadata. It's worth to note that the keys are loaded through callback functions which the platform lists in the key descriptor table.

2. Once the TF-M Crypto service is initialised, at runtime it might receive a request through an API call to use one of the builtin key IDs. Those IDs are described in the tfm_builtin_key_ids.h header. Platforms can override the default values providing their own header.
3. When the PSA Crypto core in the psa_get_and_lock_key_slot() function checks that the key_id being requested is in the builtin range region, it probes the platform through the function mbedtls_psa_platform_get_builtin_key which must be implemented by the TF-M Crypto service library abstraction layer. This function just checks the key descriptor table from the HAL to understand if such key_id is available in the platform and what are the corresponding slot and lifetime values associated to it. The lifetime contains the location of the key, which determines which driver is responsible for dealing with it (for the tfm_builtin_key_loader driver, that is TFM_BUILTIN_KEY_LOADER_KEY_LOCATION), while the slot value maps the key to the slot in the driver internal storage.
4. At this point, the PSA Crypto core knows that the key exists and is bound to the location associated to the driver. By calling into the driver wrappers layer is then able to retrieve the key attributes stored in the platform for that key ID, and the size required to allocate in its internal slot management system in order to load the key material in the core. This is done by calling tfm_builtin_key_loader_get_builtin_key just with a valid key attributes pointer (and nothing else), to retrieve the attributes. Once the attributes are available, the required size is retrieved through the driver wrapper by calling tfm_builtin_key_loader_get_key_buffer_size.
5. At this stage, the slot management subsystem calls again into the driver wrapper layer through tfm_builtin_key_loader_get_builtin_key with a valid buffer to hold the key material returned by the tfm_builtin_key_loader driver. When loading the key, the user requiring that key_id is validated by the driver code against the policies defined by the platform. If the policies match, the builtin key material and metadata is loaded and is used like a transparent key available to the PSA Crypto core slot management subsystem.

Technical details

Builtin key IDs and overriding

TF-M builtin key IDs are defined in interface/include/crypto_keys/tfm_builtin_key_ids.h through the enum tfm_key_id_builtin_t. They are allocated inside the range that PSA specifies for the builtin keys, i.e. between MBEDTLS_PSA_KEY_ID_BUILTIN_MIN and MBEDLTS_PSA_KEY_ID_BUILTIN_MAX. A platform can specify extra builtin key IDs by setting the PLATFORM_DEFAULT_CRYPTO_KEYS variable to OFF, creating the header tfm_builtin_key_ids.h, and specifying new keys and IDs.

Builtin key access control

A builtin key is accessible by all callers since the key_id associated to it is public information. Access to the keys must be mediated, which is done by matching the user requesting the key_id against the policies available for that user on that particular key in the policy table. If no policies are specified for a specific combination of user and key_id, the usage flags in the key attributes will be all set to zeros, meaning the key will be unusable for any operation for that particular user.

Multi-partition key derivation

The HUK is used for key derivation by any secure partition or NS caller that requires keys that are bound to a particular context. For example, Protected Storage derives keys uniquely for each user of the service which are used to encrypt user files. In order to provide HUK derivation to

every secure partition / NS caller, it must be ensured that no service that utilises HUK derivation can derive the same key as another service (simply by using the same inputs for the KDF APIs, i.e. accessing the same base key for derivation).

This is accomplished by deriving a further "platform key" for each builtin key that has PSA_KEY_USAGE_DERIVE set in its attributes. These platform keys are derived from the builtin key, using the partition ID as a KDF input, and can then be used for safely for further derivations by the user, without risks to derive the same keys as other users. This is enforced directly by the tfm_builtin_key_loader driver.

Note: If the NS client ID feature is disabled, all NS callers share a partition ID of -1, and therefore will share a platform key and be therefore be able to derive the same keys as other NS callers.

For keys that are not exposed outside the device, this is transparent to the service that is using the key derivation, as they have no access to the builtin key material and cannot distinguish between keys derived directly from it and keys derived from the platform key. For some builtin keys, deriving platform keys is not acceptable, as the key is used outside the device (i.e. the IAK public key is used to verify attestation tokens) so the actual builtin key is used.

The decision has been taken to derive platform keys for any key that can be used for key derivation (PSA_KEY_USAGE_DERIVE), and not derive platform keys otherwise. For builtin keys that do not derive platform keys but are directly used, care must be taken with access control where multiple partitions have access to the same raw key material.

Mbed TLS transparent builtin keys

Mbed TLS does not natively support transparent builtin keys (transparent keys are keys where the key material is directly accessible by the PSA Crypto core), so some modifications had to be made. Opaque keyslots have the same basic structure as standard transparent key slots, and can be passed to the functions usually reserved for transparent keys, though this is a private implementation detail of the Mbed TLS library and is not specified in the driver interface. Therefore, the only modification required currently is to allow keys that have the location TFM_BUILTIN_KEY_LOADER_KEY_LOCATION to be passed to the functions that usually accept transparent keys only, i.e. with the location PSA_KEY_LOCATION_LOCAL_STORAGE. This is due to the fact that the standard assumption of the PSA Crypto core is that, if a driver that provides an additional location, will also provide dedicated cryptographic mechanisms to act on those keys, but this is not the case of the tfm_builtin_key_loader, as it just provides a mechanism to load keys (which act as a transparent key with local storage, once loaded), but Mbed TLS does not support such "transparent builtin key" concept. Note that the modifications on Mbed TLS are relying on non standard implementation details hence this particular integration can change between releases[3].

Using Opaque PSA crypto accelerators with TF-M

For platforms which have a cryptographic accelerator which has a corresponding Opaque PSA crypto accelerator driver, the TF-M builtin key loader driver can be disabled using the -DCRYPTO_TFM_BUILTIN_KEYS_DRIVER=OFF cmake option. The platform can then redefine the HAL function tfm_plat_builtin_key_get_desc_table_ptr to point to a table where the location and slot number of the keys corresponds instead to the opaque driver. The PSA driver wrapper will then route the calls into the opaque driver, with no other changes needed. If the description table is altered but the builtin key loader driver is not disabled, it is possible to mix software builtin keys with keys stored in opaque accelerators on a per-key level. Note that because the key policy enforcement via tfm_plat_builtin_key_get_desc_table_ptr is currently applied by the builtin key loader driver, other opaque drivers must apply either this policy or their own policy (Though this may be changed in future).

References

[3] Interface for platform keys: https://github.com/ARM-software/psa-crypto-api/issues/550

Copyright (c) 2022-2023, Arm Limited. All rights reserved.

Log system design document

Author
Shawn Shan

Organization
Arm Limited

Contact
shawn.shan@arm.com

Background

In current TF-M log system, the SPM and Secure partitions share the same log APIs and implementations. While TF-M is keep evolving, the requirements for the log system has changed:

- Log level is required for both SPM and SP sides to output message in different scenarios.
- SPM only needs simple log format such as hex and string, while SP needs rich formatting.
- Distinctions on log output between SPM and SP are required.

A new log system is needed to separate the SPM and Secure partitions and to meet their different requirements.

Design

To allow customizable configurations, the log interfaces are defined as macros. The macros are easy to be forwarded or even empty. When SPM trying to output message and a value, it relies on a wrapper function, and finally output the formatted message by the HAL API.

The design principles of TF-M log system:

- Configurable log levels.
- Separated SPM and SP log implementations.
- Platforms provide log HAL implementations.

SPM Log System

Level Control

Three log levels for SPM log system are defined:

- TFM_SPM_LOG_LEVEL_DEBUG
- TFM_SPM_LOG_LEVEL_INFO
- TFM_SPM_LOG_LEVEL_ERROR
- TFM_SPM_LOG_LEVEL_SILENCE

Then a macro TFM_SPM_LOG_LEVEL is defined as an indicator, it should be equal to one of the four log levels.

API Definition

The following three APIs LOG APIs output the given 'msg' with hexadecimal formatted 'val' together. These APIs provide constrained ability to output numbers inside SPM. The 'msg' can be skipped with giving an empty string like "". And these APIs supports constant 'msg' string only, giving a runtime string as parameter 'msg' would potentially cause a runtime error.

SPMLOG_DBGMSGVAL(msg, val);

SPMLOG_INFMSGVAL(msg, val);

SPMLOG_ERRMSGVAL(msg, val);

A C-function needs to work as an underlayer for these APIs as string formatting is required. Check 'spm_log_msgval' for details.

```
/**
 * brief Output the given message plus one value as hexadecimal. The message
 *       can be skipped if the 'msg' is 'NULL' or 'len' equals 0. The
 *       formatted hexadecimal string for 'value' has a '0x' prefix and
 *       leading zeros are not stripped. This function rely on HAL API
 *       'tfm_hal_output_spm_log' to output the formatted string.
 *
 * \param[in]  msg    A string message
 * \param[in]  len    The length of the message
 * \param[in]  value  A value need to be output
 *
 * \retval >=0        Number of chars output.
 * \retval <0         TFM HAL error code.
 */
int32_t  spm_log_msgval(const char *msg, size_t len, uint32_t value)
```

The following three APIs output a message in string.

SPMLOG_DBGMSG(msg);

SPMLOG_INFMSG(msg);

SPMLOG_ERRMSG(msg);

Here is a table about the effective APIs with different SPM log level.

	TFM_SPM_LOG_LEV	TFM_SPM_LOG_LE	TFM_SPM_LOG_LE\	TFM_SPM_LOG_LEVEL_SILENCE
SPM-LOG_DBGMSG'	Yes	No	No	No
SPM-LOG_INFMSGV	Yes	Yes	No	No
SPM-LOG_ERRMSG\	Yes	Yes	Yes	No
SPM-LOG_DBGMSG	Yes	No	No	No
SPM-LOG_INFMSG	Yes	Yes	No	No
SPM-LOG_ERRMSG	Yes	Yes	Yes	No

HAL API

Define HAL API for SPM log system:

```
/* SPM log HAL API */
int32_t tfm_hal_output_spm_log(const char *str, uint32_t len);
```

Take debug message as an example:

```
/* For debug message */
#define SPMLOG_DBGMSG(msg) tfm_hal_output_spm_log(msg, sizeof(msg))
/* For debug message with a value */
#define SPMLOG_DBGMSGVAL(msg, val) spm_log_msgval(msg, sizeof(msg), val)
```

Partition Log System

Partition log outputting required rich formatting in particular cases. There is a customized print inside TF-M(printf), and it is wrapped as macro.

Level Control

Three log levels for partition log system are defined:

- TFM_PARTITION_LOG_LEVEL_DEBUG
- TFM_PARTITION_LOG_LEVEL_INFO
- TFM_PARTITION_LOG_LEVEL_ERROR
- TFM_PARTITION_LOG_LEVEL_SILENCE

Then a macro TFM_PARTITION_LOG_LEVEL is defined as an indicator. It should be equal to one of the four log levels and it is an overall setting for all partitions.

Log Format

Compared to SPM, SP log API supports formatting. Similar to printf, these log APIs use a format outputting to output various type of data:

```
%d - decimal signed integer
%u - decimal unsigned integer
%x - hex(hexadecimal)
%c - char(character)
%s - string
```

API Definition

Define partition log APIs:

LOG_DBGFMT(...);

LOG_INFFMT(...);

LOG_ERRFMT(...);

Here is a table about the effective APIs with different partition log level.

	TFM_PARTITION_LO(	TFM_PARTITION_LC	TFM_PARTITION_LO(	TFM_PARTITION_LOG_LEVEL_SILENC
LOG_DB(	Yes	No	No	No
LOG_INF	Yes	Yes	No	No
LOG_ERI	Yes	Yes	Yes	No

HAL API

Please refers to the HAL design document.

Log Devices

In most of the cases, a serial device could be used as a log device. And in other particular cases, a memory-based log device could be applied as well. These log device interfaces are abstracted into HAL APIs.

Note: It is not recommended to re-use the same HAL for both SPM and SP log outputting especially when SPM and SP run under different privileged level, which makes them have a different information confidential level. Unless:

- The SPM log outputting would be disabled as silence in the release version.

Copyright (c) 2020, Arm Limited. All rights reserved.

Physical attack mitigation in Trusted Firmware-M

Authors
Tamas Ban; David Hu

Organization
Arm Limited

Contact
tamas.ban@arm.com; david.hu@arm.com

Requirements

PSA Certified Level 3 Lightweight Protection Profile[1] requires protection against physical attacks. This includes protection against manipulation of the hardware and any data, undetected manipulation of memory contents, physical probing on the chip's surface. The RoT detects or prevents its operation outside the normal operating conditions (such as voltage, clock frequency, temperature, or external energy fields) where reliability and secure operation has not been proven or tested.

Note: Mitigation against certain level of physical attacks is a mandatory requirement for PSA Level 3 certification. The *TF-M countermeasures against physical attacks* discussed below doesn't provide mitigation against all the physical attacks considered in scope for PSA L3 certification. Please check the Protection Profile document for an exhaustive list of requirements.

Physical attacks

The goal of physical attacks is to alter the expected behavior of a circuit. This can be achieved by changing the device's normal operating conditions to untested operating conditions. As a result a hazard might be triggered on the circuit level, whose impact is unpredictable in advance but its effect can be observed. With frequent attempts, a weak point of the system could be identified and the attacker could gain access to the entire device. There is a wide variety of physical attacks, the following is not a comprehensive list rather just give a taste of the possibilities:

- Inject a glitch into the device power supply or clock line.
- Operate the device outside its temperature range: cool down or warm it up.
- Shoot the chip with an electromagnetic field. This can be done by passing current through a small coil close to the chip surface, no physical contact or modification of the PCB (soldering) is necessary.
- Point a laser beam on the chip surface. It could flip bits in memory or a register, but precise knowledge of chip layout and design is necessary.

The required equipment and cost of these attacks varies. There are commercial products to perform such attacks. Furthermore, they are shipped with a scripting environment, good documentation, and a lot of examples. In general, there is a ton of videos, research paper and blogs about fault injection attacks. As a result the threshold, that even non-proficient can successfully perform such attack, gets lower over time.

Effects of physical attacks in hardware and in software execution

[1] PSA Certified Level 3 Lightweight Protection Profile

The change in the behavior of the hardware and software cannot be seen in advance when performing a physical attack. On circuit-level they manifest in bit faults. These bit faults can cause varied effects in the behavior of the device micro-architecture:

- Instruction decoding pipeline is flushed.
- Altering instructions when decoding.
- Altering data when fetching or storing.
- Altering register content, and the program counter.
- Flip bits in register or memory.

These phenomenons happen at random and cannot be observed directly but the effect can be traced in software execution. On the software level the following can happen:

- A few instructions are skipped. This can lead to taking different branch than normal.
- Corrupted CPU register or data fetch could alter the result of a comparison instruction. Or change the value returned from a function.
- Corrupted data store could alter the config of peripherals.
- Very precise attacks with laser can flip bits in any register or in memory.

This is a complex domain. Faults are not well-understood. Different fault models exist but all of them target a specific aspect of fault injection. One of the most common and probably the easily applicable fault model is the instruction skip.

Mitigation against physical attacks

The applicability of these attacks highly depends on the device. Some devices are more sensitive than others. Protection is possible at hardware and software levels as well.

On the hardware level, there are chip design principles and system IPs that are resistant to fault injection attacks. These can make it harder to perform a successful attack and as a result the chip might reset or erase sensitive content. The device maker needs to consider what level of physical attack is in scope and choose a SoC accordingly.

On top of hardware-level protection, a secondary protection layer can be implemented in software. This approach is known as “defence in depth”.

Neither hardware nor software level protection is perfect because both can be bypassed. The combination of them provides the maximum level of protection. However, even when both are in place, it is not certain that they provide 100% protection against physical attacks. The best of what is to achievable to harden the system to increase the cost of a successful attack (in terms of time and equipment), thereby making it non profitable to perform them.

Software countermeasures against physical attacks

There are practical coding techniques which can be applied to harden software against fault injection attacks. They significantly decrease the probability of a successful attack:

- Control flow monitor

  To catch malicious modification of the expected control flow. When an important portion of a program is executed, a flow monitor counter is incremented. The program moves to the next stage only if the accumulated flow monitor counter is equal to an expected value.

- Default failure

  The return value variable should always contain a value indicating failure. Changing its value to success is done only under one protected flow (preferably protected by double checks).

- Complex constant

  It is hard to change a memory region or register to a pre-defined value, but usual boolean values (0 or 1) are easier to manipulate.

- Redundant variables and condition checks

  To make branch condition attack harder it is recommended to check the relevant condition twice (it is better to have a random delay between the two comparisons).

- Random delay

  Successful fault injection attacks require very precise timing. Adding random delay to the code execution makes the timing of an attack much harder.

- Loop integrity check

  To avoid to skip critical loop iterations. It can weaken the cryptographic algorithms. After a loop has executed, check the loop counter whether it indeed has the expected value.

- Duplicated execution

  Execute a critical step multiple times to prevent fault injection from skipping the step. To mitigate multiple consecutive fault injections, random delay can be inserted between duplicated executions.

These techniques should be applied in a thoughtful way. If it is applied everywhere then it can result in messy code that makes the maintenance harder. Code must be analysed and sensitive parts and critical call path must be identified. Furthermore, these techniques increase the overall code size which might be an issue on the constrained devices.

Currently, compilers are not providing any support to implement these countermeasures automatically. On the contrary, they can eliminate the protection code during optimization. As a result, the C level protection does not add any guarantee about the final behavior of the system. The effectiveness of these protections highly depends on the actual compiler and the optimization level. The compiled assembly code must be visually inspected and tested to make sure that proper countermeasures are in-place and perform as expected.

TF-M Threat Model against physical attacks

Physical attack target

A malicious actor performs physical attack against TF-M to retrieve assets from device. These assets can be sensitive data, credentials, crypto keys. These assets are protected in TF-M by proper isolation.

For example, a malicious actor can perform the following attacks:

- Reopen the debug port or hinder the closure of it then connect to the device with a debugger and dump memory.
- Bypass secure boot to replace authentic firmware with a malicious image. Then arbitrary memory can be read.
- Assuming that secure boot cannot be bypassed then an attacker can try to hinder the setup of the memory isolation hardware by TF-M *Secure Partition Manager* (SPM) and manage to execute the non-secure image in secure state. If this is achieved then still an exploitable vulnerability is needed in the non-secure code which can be used to inject and execute arbitrary code to read the assets.
- Device might contain unsigned binary blob next to the official firmware. This can be any data, not necessarily code. If an attacker manages to replace this data with arbitrary content (e.g. a NOP slide leading to a malicious code) then they can try to manipulate the program counter to jump to this area before setting up the memory isolation.

Assumptions on attacker capability

It is assumed that the attacker owns the following capabilities to perform physical attack against devices protected by TF-M.

- Has physical access to the device.
- Able to access external memory, read and possibly tamper it.
- Able to load arbitrary candidate images for firmware upgrade.
- Able to manage that bootloader tries to upgrade the arbitrary image from staging area.
- Able to inject faults on hardware level (voltage or power glitch, EM pulse, etc.) to the system.
- Precise timing of fault injection is possible once or a few times, but in general the more intervention is required for a successful attack the harder will be to succeed.

It is out of the scope of TF-M mitigation if an attacker is able to directly tamper or disclose the assets. It is assumed that an attacker has the following technical limitations.

- No knowledge of the image signing key. Not able to sign an arbitrary image.
- Not able to directly access to the chip through debug port.
- Not able to directly access internal memory.
- No knowledge of the layout of the die or the memory arrangement of the secure code, so precise attack against specific registers or memory addresses are out of scope.

Physical attack scenarios against TF-M

Based on the analysis above, a malicious actor may perform physical attacks against critical operations in *SPE* workflow and critical modules in TF-M, to indirectly gain unauthenticated accesses to assets.

Those critical operations and modules either directly access the assets or protect the assets from disclosure. Those operations and modules can include:

- Image validation in bootloader
- Isolation management in TF-M, including platform specific configuration
- Cryptographic operations
- TF-M Secure Storage operations
- PSA client permission check in TF-M

The detailed scenarios are discussed in following sections.

Physical attacks against bootloader

Physical attacks may bypass secure image validation in bootloader and a malicious image can be installed.

The countermeasures is bootloader specific implementation and out of the scope of this document. TF-M relies on MCUboot by default. MCUboot has already implemented countermeasures against fault injection attacks[3].

Physical attacks against TF-M SPM

TF-M SPM initializes and manages the isolation configuration. It also performs permission check against secure service requests from PSA clients.

Static isolation configuration

[3] MCUboot fault injection mitigation

It is TF-M SPM's responsibility to build up isolation during the initialization phase. If this is missed or not done correctly then it might be possible for non-secure code to access some secure memory area or an external device can access assets in the device through a debug port.

Therefore, hindering the setup of memory or peripheral isolation hardware is an obvious candidate for physical attacks. The initialization phase has a constant time execution (like the previous boot-up state), therefore the timing of the attack is simpler, compared to cases when secure and non-secure runtime firmware is up-and-running for a while and IRQs make timing unpredictable.

Some examples of attacking isolation configuration are shown in the list below.

- Hinder the setting of security regions. Try to execute non-secure code as secure.
- Manipulate the setting of secure regions, try to extend the non-secure regions to cover a memory area which otherwise is intended to be secure area.
- Hinder the setting of isolation boundary. In this case vulnerable ARoT code has access to all memory.
- Manipulate peripheral configuration to give access to non-secure code to a peripheral which is intended to be secure.

PSA client permission checks

TF-M SPM performs several permission checks against secure service requests from a PSA client, such as:

- Check whether the PSA client is a non-secure client or a secure client

  NS client's PSA client ID is negative. NS client is not allowed to directly access secure areas. A malicious actor can inject faults when TF-M SPM authenticates a NS client. It may manipulate TF-M to accept it as a secure client and allow the NS client to access assets.

- Memory access checks

  TF-M SPM checks whether the request has correct permission to access a secure memory area. A malicious actor can inject faults when TF-M SPM checks memory access permission. It may skip critical check steps or corrupt the check result. Thereby a malicious service request may pass TF-M memory access check and accesses assets which it is not allowed to.

The physical attacks mentioned above relies on the a malicious NS application or a vulnerable RoT service to start a malicious secure service request to access the assets. The malicious actor has to be aware of the accurate timing of dealing with the malicious request in TF-M SPM. The timing can be affected by other clients and interrupts. It should be more difficult than pure fault injection.

Dynamic isolation boundary configuration

Physical attack may affect the isolation boundary setting during TF-M context switch, especially in Isolation Level 3. For example:

- A fault injection may cause TF-M SPM to skip clear privileged state before switching in an ARoT service.
- A fault injection may cause TF-M SPM to skip updating MPU regions and therefore the next RoT service may access assets belonging to a previous one.

However, it is much more difficult to find out the accurate timing of TF-M context switch, compared to other scenarios in TF-M SPM. It also requires a vulnerable RoT service to access assets after fault injection.

Physical attacks against TF-M Crypto service

Since crypto operations are done by mbedTLS library or by a custom crypto accelerator engine and its related software driver stack, the analysis of physical attacks against crypto operations

is out-of-scope for this document. However, in general the same requirements are applicable for the crypto, to be compliant with PSA Level 3 certification. That is, it must be resistant against physical attacks. So crypto software and hardware must be hardened against side-channel and physical attacks.

Physical attacks against Secure Storage

Physical attacks against Internal Trusted Storage

Based on the assumption in *Assumptions on attacker capability*, a malicious actor is unable to directly retrieve assets via physical attacks against *Internal Trusted Storage* (ITS).

Instead, a malicious actor can inject faults into isolation configuration of ITS area in TF-M SPM to gain the access to assets stored in ITS. Refer to *Physical attacks against TF-M SPM* for details.

Physical attacks against Protected Storage

Based on the assumption in *Assumptions on attacker capability*, a malicious actor can be able to directly access external storage device. Therefore *Protected Storage* (PS) shall enable encryption and authentication by default to detect tampering with the content in external storage device.

A malicious actor can also inject faults into isolation configuration of PS and external storage device peripherals in TF-M SPM to gain the access to assets stored in PS. Refer to *Physical attacks against TF-M SPM* for details.

It is out of the scope of TF-M to fully prevent malicious actors from directly tampering with or retrieving content stored in external storage devices.

Physical attacks against platform specific implementation

Platform specific implementation includes critical TF-M HAL implementations. A malicious actor can perform physical attack against those platform specific implementations to bypass the countermeasures in TF-M common code.

Platform early initialization

TFM provides a HAL API for platforms to perform HW initialization before SPM initialization starts. The system integrator is responsible to implement this API on a particular SoC and harden it against physical attacks:

```
enum tfm_hal_status_t tfm_hal_platform_init(void);
```

The API can have several initializations on different modules. The system integrator can choose to even harden some of these initializations functions within this platform init API. One of the examples is the debug access setting.

Debug access setting

TF-M configures debug access according to device lifecycle and accessible debug certificates. In general, TF-M locks down the debug port if the device is in secure production state. The system integrator can put the settings into an API and harden it against physical attacks.

Platform specific isolation configuration

TFM SPM exposes a HAL API for static and dynamic isolation configuration. The system integrator is responsible to implement these API on a particular SoC and harden it against physical attacks.

```
enum tfm_hal_status_t tfm_hal_set_up_static_boundaries(void);
enum tfm_hal_status_t tfm_hal_bind_boundary(const struct partition_load_info_t *p_ldinf,
                                            uintptr_t *p_boundary);
```

Memory access check

TFM SPM exposes a HAL API for platform specific memory access check. The system integrator is responsible to implement this API on a particular SoC and harden it against physical attacks.

```
tfm_hal_status_t tfm_hal_memory_check(uintptr_t boundary,
                                      uintptr_t base,
                                      size_t size,
                                      uint32_t access_type);
```

TF-M countermeasures against physical attacks

This section propose a design of software countermeasures against physical attacks.

Fault injection hardening library

There is no open-source library which implements generic mitigation techniques listed in *Software countermeasures against physical attacks*. TF-M project implements a portion of these techniques. TF-M software countermeasures are implemented as a small library Fault Injection Hardening (FIH) in TF-M code base. A similar library was first introduced and tested in the MCUboot project (version 1.7.0)[2] which TF-M relies on.

The FIH library is put under TF-M lib/fih/.

The implementation of the different techniques was assigned to fault injection protection profiles. Four profiles (OFF, LOW, MEDIUM, HIGH) were introduced to fit better to the device capability (memory size, TRNG availability) and to protection requirements mandated by the device threat model. Fault injection protection profile is configurable at compile-time, default value: OFF.

Countermeasure profiles and corresponding techniques are listed in the table below.

Countermeasure	Profile LOW	Profile MEDIUM	Profile HIGH	Comments
Control flow monitor	Y	Y	Y	
Failure loop hardening	Y	Y	Y	
Complex constant		Y	Y	
Redundant variables and checks		Y	Y	
Random delay			Y	Implemented, but depends on HW capability

Similar to MCUboot, four profiles are supported. It can be configured at build time by setting (default is OFF):

-DTFM_FIH_PROFILE=<OFF, LOW, MEDIUM, HIGH>

How to use FIH library

As analyzed in *TF-M Threat Model against physical attacks*, this section focuses on integrating FIH library in TF-M SPM to mitigate physical attacks.

- Identify critical function call path which is mandatory for configuring isolation or debug access. Change their return types to FIH_RET_TYPE and make them return with FIH_RET. Then call them with FIH_CALL. These macros are providing the extra checking functionality (control flow monitor, redundant checks and variables, random delay, complex constant) according to the profile settings. More details about usage can be found here: trusted-firmware-m/lib/fih/inc/fih.h

[2] MCUboot project

Take simplified TF-M SPM initialization flow as an example:

```
main()
  |
  |--> tfm_core_init()
  |          |
  |          |--> tfm_hal_set_up_static_boundaries()
  |          |                |
  |          |                |--> platform specific isolation impl.
  |          |
  |          |--> tfm_hal_platform_init()
  |                           |
  |                           |--> platform specific init
  |
  |--> During each partition initialization
             |
             |--> tfm_hal_bind_boundary()
                             |
                             |--> platform specific peripheral isolation impl.
```

- Might make the important setting of peripheral config register redundant and verify them to match expectations before continue.
- Implements an extra verification function which checks the critical hardware config before secure code switches to non-secure. Proposed API for this purpose:

```
fih_int tfm_hal_verify_static_boundaries(void);
```

This function is intended to be called just after the static boundaries are set up and is responsible for checking all critical hardware configurations. The goal is to catch if something is missed and act according to system policy. The introduction of one more checking point requires one more intervention with precise timing. The system integrator is responsible to implement this API on a particular SoC and harden it against physical attacks. Make sure that all platform dependent security feature is properly configured.

- The most powerful mitigation technique is to add random delay to the code execution. This makes the timing of the attack much harder. However it requires an entropy source. It is recommended to use the HIGH profile when hardware support is available. There is a porting API layer to fetch random numbers in FIH library:

```
void fih_delay_init(void);
uint8_t fih_delay_random(void);
```

- Similar countermeasures can be implemented in critical steps in platform specific implementation.

  Take memory isolation settings on AN521 platform as an example. The following hardware components are responsible for memory isolation in a SoC, which is based on SSE-200 subsystem. System integrators must examine the chip specific memory isolation solution, identify the key components and harden the configuration of those. This list just serves as an example here for easier understanding:

  – Implementation Defined Attribution Unit (IDAU): Implementation defined, it can be a static config or dynamic. Contains the default security access permissions of the memory map.

  – SAU: The main module in the CPU to determine the security settings of the memory.

  – *MPC*: External module from the CPU point of view. It protects the non security aware memories from unauthenticated access. Having a properly configured MPC significantly increases the security of the system.

  – *PPC*: External module from the CPU point of view. Protects the non security aware peripherals from unauthenticated access.

  - MPU: Protects memory from unprivileged access. ARoT code has only a restricted access in secure domain. It mitigates that a vulnerable or malicious ARoT partition can access to device assets.

The following AN521 specific isolation configuration functions shall be hardened against physical attacks.

```
sau_and_idau_cfg()
mpc_init_cfg()
ppc_init_cfg()
```

Some platform specific implementation rely on platform standard device driver libraries. It can become much more difficult to maintain drivers if the standard libraries are modified with FIH library. Platform specific implementation can implement duplicated execution and redundant variables/ condition check when calling platform standard device driver libraries according to usage scenarios.

Impact on memory footprint

The addition of protection code against physical attacks increases the memory footprint. The actual increase depends on the selected profile and where the mitigation code is added.

Attack experiment with SPM

The goal is to bypass the setting of memory isolation hardware with simulated instruction skips in fast model execution (FVP_MPS2_AEMv8M) in order to execute the regular non-secure test code in secure state. This is done by identifying the configuration steps which must be bypassed to make this happen. The instruction skip simulation is achieved by breakpoints and manual manipulation of the program counter. The following steps are done on AN521 target, but this can be different on another target:

- Bypass the configuration of isolation HW: SAU, MPC.
- Bypass the setting of the PSP limit register. Otherwise, a stack overflow exception will happen. Because the secure PSP will be overwritten by the address of the non-secure stack and on this particular target the non-secure stack is on lower address than the value in the secure PSP_LIMIT register.
- Avoid the clearing of the least significant bit in the non-secure entry point, where BLXNS/BXNS is jumping to non-secure code. Having the least significant bit cleared indicates to the hardware to switch security state.

The previous steps are enough to execute the non-secure Reset_Handler() in secure state. Usually, RTOS is executing on the non-secure side. In order to properly boot it up further steps are needed:

- Set the S_VTOR system register to point the address of the NS Vector table. Code is executed in secure state therefore when an IRQ hit then the handler address is fetched from the table pointed by S_VTOR register. RTOS usually do an SVC call at start-up. If S_VTOR is not modified then SPM's SVC handler will be executed.
- TBC: RTX osKernelStart still failing.

The bottom line is that in order to execute the regular non-secure code in secure state the attacker need to interfere with the execution flow at many places. Successful attack can be made even harder by adding the described mitigation techniques and some random delays.

Reference

Copyright (c) 2021-2022, Arm Limited. All rights reserved.

Copyright (c) 2021, Arm Limited. All rights reserved.

Contribution Guidelines

Contributing Process

Contributions to the TF-M project need to follow the process below.

Note: Please contact TF-M mailing list for any question.

- It is recommended to subscribe to TF-M mailing list via this page.
- Refer to the *Roadmap* or send a mail to the TF-M mailing list to get the latest status and plan of TF-M.
- Follow *Design Proposal Guideline* to propose your design.
- Follow guidelines below to prepare the patch:
  - Clone the TF-M code on your own machine from TF-M git repository.
  - Follow the TF-M getting started, *Build Instructions Coding Guide* for the TF-M project.
  - Make your changes in logical chunks to help reviewers. Each commit should be a separate review and either work properly or be squashed after the review and before merging.
  - Follow *Documentation Contribution Guidelines* to update documentation in `docs` folder if needed.
  - Test your changes and add details to the commit description.
  - The code is accepted under *Developer Certificate of Origin (DCO)*. Use `git commit -s` to add a `Signed-off-by` trailer at the end of the commit log message. See git-commit for details.
  - Ensure that each changed file has the correct copyright and license information. Files that entirely consist of contributions to this project should have a copyright notice and BSD-3-Clause SPDX license identifier of the form as shown in *License*. Files that contain changes to imported Third Party IP files should retain their original copyright and license notices.

    Contributors can add the following copyright note, (whilst) it is suggested to update copyright note only for *major*, non-trivial changes.

    ```
    Copyright (c) XXXX[-YYYY], <OWNER>. All rights reserved.
    ```

    where XXXX is the year of first contribution and YYYY is the optional year of most recent contribution. <OWNER> is your or your company name.
  - Add a Change-Id to the commit message, which can be generated any way you like (e.g. from the SHA of the commit). It is suggested to clone repositories with commit-msg hook. The commit-msg hook attaches Change-Id automatically. Take trusted-firmware-m as an example.
- Submit your patch for review. Refer to Uploading Changes for details of uploading patch.
- Add relevant *code owner(s)* for reviewing the patch.
- You may be asked to provide further details or make additional changes.
- You can discuss further with code owner(s) and maintainer(s) directly via TF-M mailing list if necessary.

- If multiple patches are linked in a chain then code owners and maintainers should review and merge individual patches when they are ready, rather than waiting for the entire chain to be reviewed. If multiple patches are intended to be merged together then authors shall use topics or explicitly mention that in the commit message.
- Click Allow-CI +1 button on Gerrit page to run CI to validate your patch. Your patch shall pass CI successfully before being merged. Code owner(s) and maintainer(s) may ask for additional test.
- Once the change is approved by code owners, the patch will be merged by the maintainer.

Copyright (c) 2017-2022, Arm Limited. All rights reserved.

Code Review Guideline

The purpose of this document is to clarify design items to be reviewed during the code review process.

Please contact *maintainers* or write an e-mail thread on the TF-M mailing list for any questions.

List of the guidelines

The prerequisites before going to the review stage:

- Read the *Contributing Process* to know basic concepts.
- Read the *Source Structure* for structure related reference.

The review guidelines consist of these items:

- *Exceptions*.
- *Non-source items*.
- *Namespace*.
- *Assembler code*.
- *Include*.
- *Auto-doc*.
- *Commit Message*.

Note: Each guideline item is assigned with a unique symbol in the format Rx.y, while x and y are numeric symbols. These symbols are created for easy referencing during the code review process.

Exceptions

Files under listed folders are fully or partially imported from 3rd party projects, these files follow the original project defined guidelines and styles:

Important:

- R1.1 ext and its subfolders.

Non-source items

For all non-source files such as build system files or configuration files:

Important:

- R2.1 Follow the existing style while making changes.

Namespace

TF-M assign the namespace to files and symbols for an easier code reading. The symbol here includes functions, variables, types and MACROs. The prerequisites:

Important:

- R3.1 Follow the documents or specifications if they propose namespaces ('psa_' for PSA defined items E.g.,). Ask the contributor to tell the documents or specifications if the reviewer is not sure about a namespace.
- R3.2 Assign TF-M specific namespace 'tfm_' or 'TFM_' for symbols implementing TF-M specific features. Ask the contributor to clarify the purpose of the patch if the reviewer is not sure about the namespace.

For the sources out of the above prerequisites (R3.1 and R3.2):

Important:

- R3.3 Do not assign a namespace for source files.
- R3.4 Assigning a namespace for interface symbols is recommended. The namespace needs to be expressed in one word to be followed by other words into a phrase can represent the functionalities implemented. One word naming is allowed if the namespace can represent the functionality perfectly.
- R3.5 Assign a namespace for private symbols is NOT recommended.
- R3.6 Do not assign characters out of alphabets as the leading character.

Examples:

```
/* R3.1 FILE: s/spm/core/psa_client.c */

/* R3.2 FILE: s/spm/core/tfm_secure_context.c */

/* R3.4 FILE: s/spm/core/spm.c, 'spm\_' as the namespace */
void spm_init(void);

/* R3.5 FILE: s/spm/core/main.c */
static void init_functions(void);

/* R3.6 Not permitted: */
/* static uint32_t ___count; */
```

Assembler code

Important:

- R4.1 Pure assembler sources or inline assembler code are required to be put under the platform-independent or architecture-independent folders. The logic folders should not contain any assembler code, referring to external MACRO wrapped assembler code is allowed. Here is one example of the logic folder:
  - 'secure_fw/spm'.

Examples:

```
/*
 * R4.1 The following MACRO is allowed to be referenced under
 * 'secure_fw/spm'
 */
#define SVC(code) __asm volatile("svc  %0", ::"I"(code))
```

Include

This chapter describes the placement of the headers and including. There are two types of headers: The interface headers contain symbols to be shared between modules and the private headers contain symbols only for internal usage.

Important:

- R5.1 Put the interface header of one module in the include folder under the root of this module. Deeper sub-folders can not have include folders, which means only one include is allowed for one module.
- R5.2 Creating sub-folders under include to represent the more granular scope of the interfaces is allowed.
- R5.3 private header can be put at the same place with the implementation sources for the private symbols contained in the header. It also can be put at the place where the sources need it. The latter is useful when some "private header" contains abstracted interfaces, but these interfaces are not public interfaces so it won't be put under "include" folder.
- R5.4 Use <> when including public headers.
- R5.5 Use "" when including private headers.
- R5.6 The module's include folder needs to be added into referencing module's header searching path.
- R5.7 The module's include folder and the root folder needs to be added into its own header searching path and apply a hierarchy including with folder name.
- R5.8 Path hierarchy including is allowed since there are sub-folders under include folder and the module folder.
- R5.9 The including statement group order: the beginning group contains toolchain headers, then follows the public headers group and finally the private headers group.
- R5.10 The including statement order inside a group: Compare the include statement as strings and sort by the string comparison result.
- R5.11 The header for the referenced symbol or definition must be included even this header is included inside one of the existing included headers. This improves portability in case the existing header implementation changed.

Examples:

```
/*
 * The structure:
 *   module_a/include/func1.h
 *   module_a/include/func2/fmain.h
 *   module_a/func1.c
 *   module_a/func2/fmain.c
 *   module_b/include/funcx.h
 *   module_b/include/funcy/fmain.h
```

(continues on next page)

(continued from previous page)

```
 *   module_b/funcx.c
 *   module_b/funcxi.h
 *   module_b/funcy/fmain.c
 *   module_b/funcy/fsub.c
 *   module_b/funcy/fsub.h
 * Here takes module_b/funcx.c as example:
 */
#include <func1.h>
#include <func2/fmain.h>
#include <funcx.h>
#include "funcxi.h"
#include "funcy/fsub.h"
```

Auto-doc

Auto document system such as doxygen is useful for describing interfaces. While it would be a development burden since the details are described in the design documents already. The guidelines for auto-doc:

Important:

- R6.1 Headers and sources under these folders need to apply auto-doc style comments: *include.
- R6.2 Developers decide the comment style for sources out of listed folders.

Commit Message

TF-M has the requirements on commit message:

Important:

- R7.1 Assign correct topic for a patch. Check the following table.

Topic	Justification
Boot	*bl2/** or *bl1/**
Build	For build system related purpose.
Docs	All *.rst changes.
Dualcpu	Dual-cpu related changes.
HAL	Generic HAL interface/implementation changes.
Interface	Interface changes, either Non-source and secure.
Pack	For packing purpose.
Platform	Generic platform related changes under *platform/**.
Platform Name	Specific platform changes.
Partition	Multiple partition related changes.
Partition Name	Specific partition related changes.
Service	Multiple service related changes.
Service Name	Specific service related changes.
SPM	*secure_fw/spm/**
SPRTL	*secure-fw/partitions/lib/runtime/**
Tool	*tools* folder or *tf-m-tools* repo

Note: Ideally, one topic should cover one specific type of changes. For crossing topic changes, check the main part of the change and use the main part related topic as patch topic. If there is

no suitable topics to cover the change, contact the community for an update.

Copyright (c) 2020-2022, Arm Limited. All rights reserved.

Trusted Firmware-M (TF-M) is an open govcommunity project. All contributions are ultimately merged by the maintainers listed below. Technical ownership of most parts of the codebase falls on the code owners listed below. An acknowledgement from these code owners is required before the maintainers merge a contribution.

More details may be found in the Project Maintenance Process document.

Maintainers

Anton Komlev

email
Anton.Komlev@arm.com

github
Anton-TF

Antonio de Angelis

email
Antonio.deAngelis@arm.com

github
adeaarm

Chris Brand

email
Chris.Brand@cypress.com

github
UEWBot

David Hu

email
David.Hu2@arm.com

github
davidhuziji

Code owners

Bootloader

Tamas Ban

email
Tamas.Ban@arm.com

github
tamasban

David Vincze

email
David.Vincze@arm.com

github
davidvincze

Firmware Update (FWU)

Maulik Patel

email
Maulik.Patel@arm.com

github
maulik-arm

BL1 immutable bootloader

Raef Coles

email
Raef.Coles@arm.com

github
RcColes

Secure Storage

Matthew Dalzell

email
Matthew.Dalzell@arm.com

github
mdalzellarm

Crypto

Antonio de Angelis

email
Antonio.deAngelis@arm.com

github
adeaarm

David Vincze

email
David.Vincze@arm.com

github
davidvincze

Framework (SPM, etc.)

Nicola Mazzucato

email
Nicola.Mazzucato@arm.com

github
nicola-mazzucato-arm

Attestation

Maulik Patel

email
Maulik.Patel@arm.com

github
maulik-arm

Platform Partition

Nicola Mazzucato

email
Nicola.Mazzucato@arm.com

github
nicola-mazzucato-arm

Build System

Raef Coles

email
Raef.Coles@arm.com

github
RcColes

Anton Komlev

email
Anton.Komlev@arm.com

github
Anton-TF

Tests

Matthew Dalzell

email
Matthew.Dalzell@arm.com

github
mdalzellarm

SCMI Partition (TF-M-Extras)

Nicola Mazzucato

email
Nicola.Mazzucato@arm.com

github
nicola-mazzucato-arm

Arm Platforms

MPS2, MPS3, MPS4, Musca(B1,S1)

David Hazi

email
David.Hazi@arm.com

github
david-hazi-arm

Corstone1000

Hugues Kamba Mpiana

email
Hugues.KambaMpiana@arm.com

github
hugueskamba

Emekcan Aras

email
Emekcan.Aras@arm.com

github
ememarar

RSE

Raef Coles

email
Raef.Coles@arm.com

github
RcColes

Antonio de Angelis

email
Antonio.deAngelis@arm.com

github
adeaarm

Arm Automotive RD

Diego Sueiro

email
diego.sueiro@arm.com

github
diego-sueiro

Peter Hoyes

email
peter.hoyes@arm.com

github
hoyes

Ziad Elhanafy

email
ziad.elhanafy@arm.com

github
ZiadElhanafy

NXP Platforms

Ruchika Gupta

email
ruchika.gupta_1@nxp.com

github
ruchi393

STM Platforms

Ahmad EL JOUAID

email
ahmad.eljouaid@st.com

github
ahmadstm

Infineon/Cypress Platforms

Chris Brand

email
Chris Brand@cypress.com

github
UEWBot

Laird Connectivity Platforms

Greg Leach

email
Greg.Leach@lairdconnect.com

github
greg-leach

Nordic Semiconductor Platforms

Georgios Vasilakis

email
georgios.vasilakis@nordicsemi.no

github
Vge0rge

Nuvoton Platform

WS Chang

email
MS20 WSChang0@nuvoton.com

github
wschang0

ArmChina Platform

Jidong Mei

email
Jidong.Mei@armchina.com

github
JidongMei

Raspberry Pi Platform

William Vinnicombe

email
William.Vinnicombe@raspberrypi.com

github
Raspberry Pi

Analog Devices Platform

Sadik Ozer

email
Sadik.Ozer@analog.com

github
ozersa

SPDX-License-Identifier: BSD-3-Clause

SPDX-FileCopyrightText: Copyright The TrustedFirmware-M Contributors

Yet another coding standard :)

Warning: Every rule has an exception so if you disagree or dislike then reach out!

The coding style of TF-M project is based on Linux coding style but there are updates for domain specific conventions as listed below.

TF-M also reuses code from other SW projects, e.g. CMSIS_5, which means some areas of code may have different styles. We use common sense approach and new code may inherit coding style from external projects but it needs to remain within clear scope.

The guidance below is provided as a help. It isn't meant to be a definitive list.

As implied in the *contributing guide* maintainers have the right to decide on what's acceptable in case of any divergence.

Warning: Text files do not fall within these rules as they may require different formatting.

Consistent style

The code needs to be consistent with itself, so if existing code in the file violates listed coding style rules then it is better to follow existing style in the file and not break consistency by following the rules listed here.

You may need to add a comment in the commit description to clarify this.

List of rules

- Use 4 spaces indentation. No tabs.
- Use lower_case_with_underscore for filenames, variable and function names.
- Use standard types e.g. uint32_t, uint16_t, uint8_t, int32_t etc.
- Use uintptr_t type when representing addresses as numbers.
- Use static for all private functions and variables.
- Use void argument if your function doesn't contain any argument.
- Use C90 /* */ for comments rather than C99 //
- No trailing spaces in code.
- Keep up to 100 characters per line but 140 chars can be accepted exceptionally.
- Open curly brace { at the same if/else/while/for/switch statement line.
- Use curly braces { } for one line statement bodies also.

- Put open curly brace in the line below the function header.
- Order function parameters so that input params are before output params.
- Declare local variables at the beginning of the function.
- Define macros in all caps i.e. CAPITAL_WITH_UNDERSCORE.
- Use typedefs **ONLY** for function prototype declarations.
- Type definitions in lower_case_with_underscore ended by _t.
- Do not use typedef for other constructs as it obfuscates code.
- Do not use bitfields.
- Use static for all global variables to reduce the variable scope.
- Use enumeration for error codes to keep the code readable.
- Use descriptive variable and functions names.
- Put the correct license header at the beginning of the file.
- Keep the files (.h/.c) self-contained, i.e. put include dependencies in the file.
- Put appropriate header define guard in .h files: ___HEADER_NAME___. Any divergence from this rules should be clearly documented.
- In a .c file, #include it's own header file first.
- Document all the public functions in the header file only.
- Document all the private functions in the source file only.
- Add "extern C " definition for C++ support in the header files.
- In the switch statement, aligned cases with the switch keyword.
- For enums, use upper case letters with digits and underscore only.
- Do not code while eating.

Copyright (c) 2018-2022, Arm Limited. All rights reserved.

Documentation Contribution Guidelines

The Trusted Firmware-M project uses Sphinx to generate the Official Documentation from Restructed Text *.rst* source files,

The aim is to align as much as possible with the official Python Documentation Guidelines while keeping the consistency with the already existing files.

The guidance below is provided as a help. It is not meant to be a definitive list.

Overview

The following short-list provides a quick summary of the rules.

- If the patch modifies a present file, the file's style should be followed
- If creating a new file, *integration guide* can be used as a reference.
- When a new style is to be expressed, consult the Python Documentation Guidelines

List of rules

The following rules should be considered:

1. H1 document title headers should be expressed by # with over-line (rows on top and bottom) of the text
2. Only ONE H1 header should allowed per document, ideally placed on top of the page.
3. H2 headers should be expressed by * with over-line
4. H2 header's text should be UNIQUE in per document basis
5. H3 headers should be expressed by an underline of '='
6. H4 headers should be expressed by an underline of '-'
7. H3 and H4 headers have no limitation about naming. They can have similar names on the same document, as long as they have different parents.
8. H5 headers should be expressed by an underline of '^'
9. H5 headers will be rendered in document body but not in menus on the navigation bar
10. Do not use more than 5 levels of heading
11. When writing guides, which are expected to be able to be readable by command line tools, it would be best practice to add long complicated tables, and UML diagrams in the bottom of the page and using internal references(auto-label)
12. No special formatting should be allowed in Headers (code, underline, strike-through etc)
13. Long URLs and external references should be placed at the bottom of the page and referenced in the body of the document
14. New introduced terms and abbreviations should be added to Glossary and directly linked by the *:term:* notation across all documents using it.

Platform Documentation

The Documentation Build system provides an interface with the platform directory allowing maintainers to bundle platform specific documentation. Platforms are grouped by vendor. **This behaviour needs to be explicitly enabled for each vendor's space** by providing the *<vendor>/index.rst* (responsible for generating the *Platform Index File*) and adding a table of contents entry for the corresponding vendor's space. The format and structure of this entry is not strictly defined, and allows flexible control of vendor's and platform's documentation space. Follow the *Platform Documentation* document for more details.

Common Use Cases

The section below describes with examples, a rule compliant implementation for most common documentation elements.

Headers

```
####################
Document Title (H1)
####################

**********
Chapter Title (H2)
**********

Chapter Section (H3)
====================

Chapter Sub-Section (H4)
------------------------
```

(continues on next page)

(continued from previous page)

```
Subsection of Chapter sub-section (H5)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Code Blocks

The recommendation for code content, is to use the explicit code-block directive, ideally with a defined lexer for the language the block contains.

A list of compatible lexers can be found at Pygments Lexers

```
.. code-block:: bash

   ls
   pwd

.. code-block:: doscon

   dir

.. code-block:: c

   static struct rn_object_t;

.. code-block:: python3

   print(”Hello TF-M”)
```

Restructured Text supports implicit code-blocks by indenting a section of text, surrounded by new lines. While this formatting is allowed, it should be avoided if possible.

```
The quick brown fox jumps over the lazy dog

   ls
   pwd
```

Note: Mixing two different code-block formats in the same document will break the whole document's rendering. When editing an existing document, please follow the existing format.

New documents should always use the explicit format.

Tables

For adding new tables the *table::* notation should be used.

```
.. table:: Components table
   :widths: auto

   +-------------+-------------+------------+
   | **Title A** | **Title B** | **Title C** |
   +=============+=============+============+
   | Text A      | Text B      | Text C     |
   +-------------+-------------+------------+
```

While the equivalent simple table code will render correctly in the output, it will not be added to the index (So it cannot be referenced if needed)

```
+-------------+-------------+------------+
| **Title A** | **Title B** | **Title C** |
+=============+=============+============+
```

(continues on next page)

(continued from previous page)

```
| Text A       | Text B       | Text C      |
+--------------+--------------+-------------+
```

Other types of tables such as list-tables and csv-tables are also permitted, as seen on /getting_started/tfm_getting_started and /releases/1.0

External Links

External links should be placed in the bottom of a document.

```
The quick brown fox jumps over the lazy dog according to `Link_A`_

.. _Link_A: https://www.aaa.org
.. _Link_B: https://www.bbb.org

--------------

*Copyright (c) XYZ *
```

Creating in-line links is permitted, but should be avoided if possible. It should be only used for consistency purposes or for a small ammount of short links.

```
The quick brown fox jumps over the lazy dog according to `Link_A <https://www.aaa.org>`_
```

If for the purposes of content, a link is to be referenced by multiple labels, internal linking is the recommended approach.

```
The quick brown fox jumps over the lazy dog according to `Link_A_New`_

.. _Link_A: https://www.aaa.org
.. _Link_A_New: `Link_A`_

--------------

*Copyright (c) XYZ *
```

Document Links

A document included in the documentation can be referenced by the *doc:* notation

```
:doc:`integration guide </integration_guide/tfm_integration_guide>`
```

The path is relative to the root of the Trusted Firmware-M code.

Trusted Firmware-M project is spread among multiple repositories: Trusted Firmware-M, TF-M Tests, TF-M Tools and TF-M Extras. Every repository has its own documentation, and since *v2.0.0*, they can be found under *Links*.

Using Intersphinx, it is now possible to use cross-reference roles from Sphinx to reference documentation from different projects (repositories), like TF-M Tests. Referencing documentation using the *doc:* notation is preferred and helps to avoid broken cross-references if the link of the document changes.

For example, to get this: Adding TF-M Regression Test Suite; the Restructed Text would look like this:

```
:doc:`Adding TF-M Regression Test Suite <TF-M-Tests:tfm_test_suites_addition>`
```

As can be seen, it is quite similar to cross-referencing in Sphinx, except the path to the document is preceded with the external project name. For TF-M Tests, it is TF-M-Tests. The names of other

projects configured to be referenced using Intersphinx can be seen in the conf.py file under intersphinx_mapping.

Reference specific section of a document

In order to reference a specific section of a document, up to level 4 headers (if they are included in the index), the *ref:* keyword can be used

```
:ref:`docs/getting_started/tfm_getting_started:Tool & Dependency overview`
```

This can also be used to quickly scroll to the specific section of the current document. This technique can be used to add complex table in the bottom of a document and create clickable quick access references to it for improved user experience.

Glossary term

For technical terms and abbreviations, the recommended guidance is to add an entry to the *Glossary of terms and abbreviations* and refer to it, using the *term:* directive

```
HAL
Hardware Abstraction Layer
   Interface to abstract hardware-oriented operations and provides a set of
   APIs to the upper layers.

.....

As described in the design document :term:`HAL` abstracts the
hardware-oriented and platform specific
.......
```

Note: The ":term:" directive does not work when used in special formatting. Using *:term:*HAL*** **will not link to the glossary term**.

References

1. Sphinx
2. Restructed Text
3. Python Documentation Guidelines
4. Pygments Lexers
5. Intersphinx

Copyright (c) 2020-2023, Arm Limited. All rights reserved.

Design proposal guideline

The design proposal guideline specifies the steps to propose and upload design proposals to TF-M. Those steps are lightweight and flexible to make sure that contributors can focus more on actual code implementation and iteration.

The guideline encourages developers to share design proposal via TF-M mailing list[1] and TF-M technical forum (tech forum)[2]. The design details can be discussed via code reviews of actual implementations.

Typical steps are shown as the diagram below.

[1] TF-M mailing list
[2] TF-M technical forum

```
@startuml

title Design proposal process

[*] --> propose : Non-trivial changes

state "Propose general ideas" as propose {
  state "TF-M mailing list" as mail_list : Contributors send emails to mailing list\nto describe the design.
  state "TF-M tech forum" as tech_forum : Contributors present the design\nin tech forum.

  [*] --> mail_list
  [*] --> tech_forum
}

note bottom of propose : Optional but strongly recommended

[*] --> upload
note right of upload : No prerequisites
propose --> upload

state "Upstream changes" as upload : Contributors upstream code patch\nand integration guide to gerrit.
state "Code review" as review : Reviewer review changes of\ncode and documents.\nChanges pass␣
↪verifications.
state "Broadcast patches" as broadcast : Contributors ask for review\nin mailing list.
state "Approve and merge" as approve : Code owners approve changes.\nMaintainers merge patches.

upload --> broadcast : Optional
upload --> review

broadcast --> review
review --> review : Update implementation
review --> approve
approve --> [*]

@enduml
```

Discussion in TF-M mailing list and tech forum

It is **highly recommended** to propose and discuss designs in TF-M mailing list or TF-M tech forum, before or while the code implementation is under review.

It is efficient and flexible to directly discuss design proposal via TF-M mailing list and TF-M tech forum. Contributors can receive quick and broad feedback from TF-M community.

Although it is optional to present the ideas in mailing list or tech forum, it will help reviewers understand the design much better and expedite the code review process.

Code review of details

It is straightforward and convenient for contributors and reviewers to deliberate over design and implementation details via code review.

Contributors can implement their design proposal and upstream the patch set to TF-M gerrit[3] for code review. For non-trivial changes or new major features, it is **strongly suggested** to propose the design to TF-M mailing list and tech forum in advance.

Contributors don't have to wait for any approvals before upstreaming patches, even if the changes are non-trivial. No formal design document in advance is required anymore.

The review process is the same as the general one[4], with some specific requirements:

[3] TF-M gerrit
[4] *Contributing process*

- Contributors can send an email to TF-M mailing list to ask for review.
- If it requires additional reviewers besides code owners and maintainers, contributors shall add the specific reviewers in the review list.
- Authors shall clearly specify the design purpose and briefly describe the implementation in the commit message.
- Authors shall put essential comments and notes in code for the code changes.

Code owners and maintainers may require contributors to further verify the implementation besides normal per-patch CI test. Contributors shall provide the verification results as requested.

Integration guide and manual

Contributors can create an integration guide or a user manual to describe how to integrate the new features related to the design proposal.

Contributors shall update the corresponding documents if the design changes existing implementation.

Reference

Copyright (c) 2022, Arm Limited. All rights reserved.

Copyright (c) 2020-2021, Arm Limited. All rights reserved.

License

Copyright (c) 2017-2019, Arm Limited. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of ARM nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

—

Note: Individual files contain the following tag instead of the full license text.

SPDX-License-Identifier: BSD-3-Clause

This enables machine processing of license information based on the SPDX License Identifiers that are here available: http://spdx.org/licenses/

Copyright (c) 2019, Arm Limited. All rights reserved.

Developer Certificate of Origin

Developer Certificate of Origin
Version 1.1

Copyright (C) 2004, 2006 The Linux Foundation and its contributors.
1 Letterman Drive
Suite D4700
San Francisco, CA, 94129

Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.

Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

(a) The contribution was created in whole or in part by me and I
have the right to submit it under the open source license
indicated in the file; or

(b) The contribution is based upon previous work that, to the best
of my knowledge, is covered under an appropriate open source
license and I have the right under that license to submit that
work with modifications, whether created in whole or in part
by me, under the same open source license (unless I am
permitted to submit under a different license), as indicated
in the file; or

(c) The contribution was provided directly to me by some other
person who certified (a), (b) or (c) and I have not modified
it.

(d) I understand and agree that this project and the contribution
are public and that a record of the contribution (including all
personal information I submit with it, including my sign-off) is
maintained indefinitely and may be redistributed consistent with
this project or the open source license(s) involved.

Copyright (c) 2019, Arm Limited. All rights reserved.

Copyright (c) 2017-2022, Arm Limited. All rights reserved.

TF-M-Tests Documents TF-M Tests

Trusted Firmware-M Tests

The Trusted Firmware-M(TF-M) Tests repo is meant to hold various tests for the Trusted Firmware-M. The TF-M tests mainly focus on functionalities of various TF-M componentes such as the TF-M SPM and various Secure Partitions.

The TF-M eRPC Test Framework

The TF-M eRPC Test Framework is an Remote Procedure Call (RPC) framework for testing purpose. It's based on the eRPC system. It is an additional framework to the existing one for running NS test suites. It enables you to run test codes on host machines as if they were running on the device. It has the following advantages.

- Off-load test codes from device to host

  Arm M-profile devices usually have limited flash storage which can only fit limited test suites. With test codes running on hosts you can run far more tests than on the devices.

- Less frequent image downloading for test code development.

  As the test codes run on the host, you don't need to update the image on device when you update test codes.

- Host can get test pass or failure directly from the return codes.

  This would be helpful for test automation because you don't need to parse the test logs.

How Does It Work

Originally, the NS tests are executed in the NSPE of the device. The NS image on the device contains the test codes, which calls into TF-M through the PSA client APIs.

With the eRPC test framework, the NS tests code are executed on the host machine. The NSPE of device side does not run the test codes anymore. When the tests call the PSA client APIs, they call into the eRPC framework. The eRPC framework communicates with the NSPE on the device, which calls into TF-M through the PSA client APIs.

The prototypes of the PSA client APIs are the same while the implementations are different. Refer to the following sections for more details.

The Stucture

The following diagram shows the software structure.

- eRPC Framework

  The eRPC framework system

- eRPC Client Shim

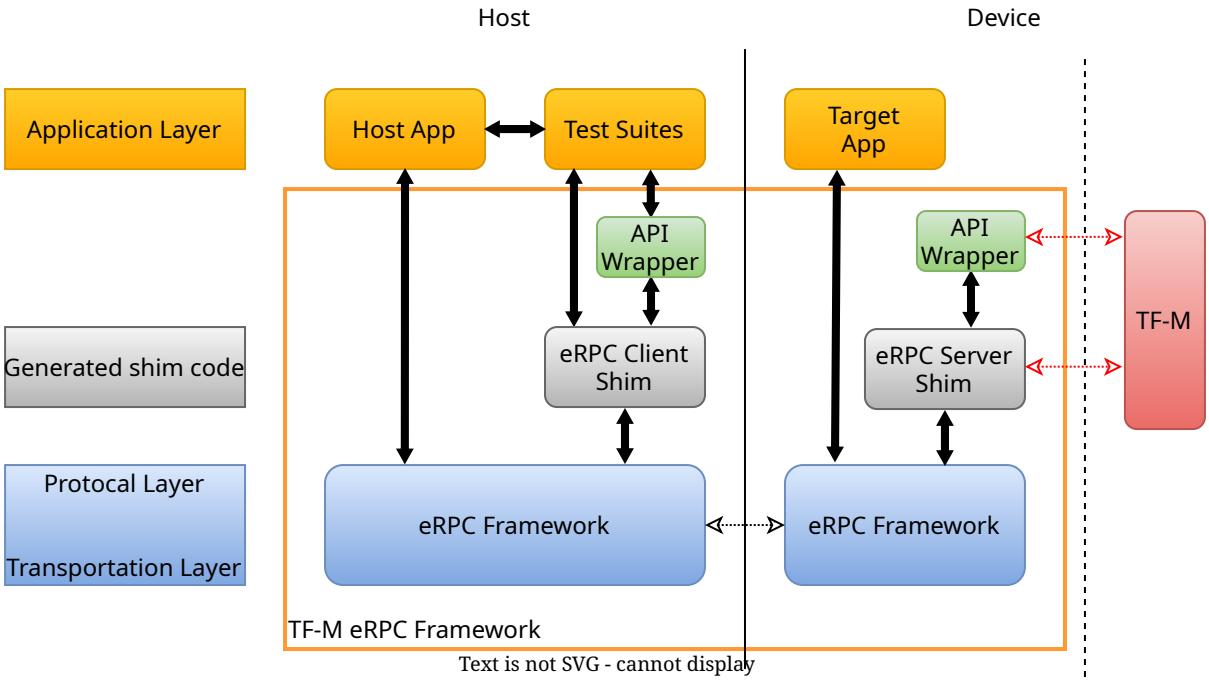
  The eRPC generated shim layer of remote APIs for clients. It serializes the identifier of the API and its parameters into a stream of bytes and transports to the server through a communication channel such as UART and TCP/IP. The codes are generated by the erpcgen tool.

- eRPC Server Shim

  The generated shim layer of the server. It registers a callback function to the eRPC framework. When the framework receives any requests from the client, it calls the callback function. The callback unserializes the bytes streams to determine what API to call and then invoke it with the corresponding parameters from the bytes streams. And then it returns the results to the client in the reverse routine.

- API Wrapper

  Part of the parameters of $\mathrm{psa_call}$ API is not supported by eRPC directly, thus an API wrapper is required to transform the $\mathrm{in_vec/out_vec}$ structures to the eRPC supported data types. The wrapper API is named as $\mathrm{erpc_psa_call}$.



On the client side, the wrapper implements the psa_call which calls the erpc_psa_call in the client shim layer. On the server side, the wrapper implements the erpc_psa_call which is called by the shim layer. The erpc_psa_call then calls the psa_call.

- Test Suites

  Can be the existing TF-M regression tests or any other tests that interact with TF-M using the PSA Client APIs.

- Host App

  Initializes the eRPC client and starts test suites.

- Target App

  Initializes the eRPC server and listens for requests from the client.

Supported APIs

The APIs supported for doing RPC are the PSA Client APIs because they are the lowest level APIs that interact with TF-M. You can build lots of test suites upon them. You can also add your own APIs in the tfm.erpc file. Please refer to IDL Reference for the syntax of the file.

API Grouping

PSA client APIs are categorised into common APIs and connection-based service APIs. Connection-based APIs are available when there are connection-based services in the TF-M. So in the eRPC integration, the APIs are also split into two groups so that the shim layer of the APIs can be separated into different files as well. Then build systems can decide which source files to build based on the existence of connection-based services.

Common APIs:

- psa_framework_version()
- psa_version()
- psa_call()

Connection-based specific:

- psa_connect()
- psa_close()

Transportation

On device side, only UART transportation is supported in NSPE. For the host side, both UART and TCP are supported. The TCP transportation support is basically for fast models where UART data are transferred between a TCP/IP socket on the host and a serial port on the target. See the fast model reference guide for more details.

Platform Integration

First, the UART drivers of platforms shall support TX/RX control feature. The TF-M build system provides a CONFIG_ENABLE_NS_UART_TX_RX_CONTROL option to enable or disable the TX/RX control feature and it is disabled by default. When the eRPC test framework is enabled, the CONFIG_ENABLE_NS_UART_TX_RX_CONTROL will be enabled automatically.

Secondly, platforms need to create their folders under the erpc/platform and then create the config_erpc_target.h to specify the UART port drivers for eRPC transportation.

```
#define ERPC_UART          Driver_USART0
```

Note: The folder structure in erpc/platform must be the same as the platform/ext/target of TF-M repo.

It's recommended to use a different UART to the stdio UART. If the same UART is used for both, then the TF-M logs (both SPM and Secure Partitions) must be disabled. Otherwise, the eRPC transportation might fail.

Application Integration

The TF-M eRPC test framework provides two CMake libraries for integration. One is the erpc_client, the other is the erpc_server. Both include the eRPC framework, the shim layers, API wrappers and expose an initialization API for client and server respectively.

The initialization does not include the initialization of the transportation layer because it is use case specific which kind of transportation is used. So it is the client and server's responsibilities to initialize the transportation layers and pass them to the erpc_client and erpc_server.

TF-M provides a app/erpc_app.c as the default server application which initializes the UART transportation and starts the eRPC server.

A config option CONFIG_TFM_ERPC_TEST_FRAMEWORK is provided to enable the eRPC framework on device (server) side. The default server will be built and developers only need to focus on the client application developments.

In summary, on the server side, you only need to build with the CONFIG_TFM_ERPC_TEST_FRAMEWORK enabled. On the client side, you must

- Initializes the transportation layer using eRPC provided APIs.
- Call the initialization function provided by TF-M eRPC test framework with the transportation instance initialized above.
- Develop the application code
- Building with CMake
  - add_subdirectory with the erpc/client
  - link the erpc_client library

There is an example at erpc/host_example for reference.

Copyright (c) 2023, Arm Limited. All rights reserved.

Adding partitions for regression tests

Introduction

Some test group may need specific test services. These test services may support one or more groups thus developers shall determine the proper test scope. Currently, TF-M test services are located under tf-m-tests/test/secure_fw.

Folder name	Description
test/secure_fw/suites/<suite>/service	Test service divided into corresponding suite subdirectories.
test/secure_fw/common_test_service	Common test services.

Implementation

Adding a test partition to provide test services is same as adding a secure partition, generally the process can be referenced from the document Adding Secure Partition.

Test Partition Specific Manifest Attributes

Each test service must have resource requirements declared in a manifest file, the contents of test services are the same as secure partitions, but their locations are different. Test service manifests shall be set in tf-m-tests/test/secure_fw/tfm_test_manifest_list.yaml.

There are some test purpose attributes in Secure Partition manifests that are **NOT** compatible with FF-M. They should be used in Test Partitions only.

weak_dependencies

A TF-M regression test Partition calls other RoT services for test. But it can still run other tests if some of the RoT services are disabled. TF-M defines a "weak_dependencies" attribute in partition manifests of regression test partitions to describe test service access to other RoT services. It *shall* be only used for TF-M regression test services.

model

A TF-M regression test Partition may support both the SFN and IPC model. The actual model being used follows the SPM backend enabled.

The TF-M build system supports this by allowing Secure Partitions to set the model attribute to dual. The manifest tool will then change it to the corresponding value according to the current backend selected.

The Test Partitions use the following definitions to know what model is being built:

- <<partition_name>>_MODEL_IPC, 1 if IPC model is used.
- <<partition_name>>_MODEL_SFN, 1 if SFN model is used.

Test service implementation

Test service of individual test

An individual test dedicated test service should be put under the corresponding test folder test/secure_fw/suites/<test_name>.

add_subdirectory(suites/<test_name>/<service_dir>) shall be added into tf-m-tests/test/secure_fw/secure_tests.cmake to make sure that the test service is built with secure side configuration.

Common test service

If a new test service is required by multiple test suites, the code should be put under test/secure_fw/common_test_services. If the new test suite relies on a common test service, please make sure that the build implementation of the test service is linked correctly, including the header files and libraries.

Copyright (c) 2022, Arm Limited. All rights reserved.

Adding TF-M Regression Test Suite

Table of Contents

- *Adding TF-M Regression Test Suite*
  - *Introduction of TF-M regression tests*
    - *Source structure*
  - *Adding a new test suite*
    - *Source code*
    - *Creating test configurations*
      - *Naming test configurations*
      - *Setting test configurations*
      - *Checking test configurations*
    - *Export TF-M configurations*
    - *Applying test configurations*
  - *Adding a new test case in test suite*
  - *Adding test services*
  - *Out-of-tree regression test suites*
    - *Example usage*
    - *Coding instructions*
      - *Header Files*
      - *Source code*
      - *CMakeLists.txt*
      - *ns_test_config.cmake*

Introduction of TF-M regression tests

TF-M regression tests test whether changes to TF-M code work as expected. A new regression test can consist of following 3 components:

1. test suite: A series of tests of a certain function.

2. test case: A specific test instance in test suites.
3. test service or partition: Optional secure services or partitions to support related test suites.

Source structure

Folder name	Description
test/bl1	TF-M bl1 test suites source code.
test/bl2	TF-M bL2 test suites source code.
test/config	The CMAKE test configurations files.
test/framework	Source code for test framework code, managing test suites.
test/secure_fw/suites	Test suites divided into subdirectories.
test/secure_fw/suites/<suite>/service	Test service divided into corresponding suite subdirectories.
test/secure_fw/common_test_service	Common test services.

Adding a new test suite

This section introduces how to add a new test suite and control its compilation with a test configuration in tests_reg/test repository.

Source code

The test suite example subdirectory named <test_name> is located under the path tests_reg/test/secure_fw/suites. If the new test suite includes both non-secure and secure parts, the source code shall be divided shared code and specific code. An example test suite folder can be organized as the figure below.

```
.
   <test_name>_tests_common.c
   non_secure
|      CMakeLists.txt
       <test_name>_ns_interface_testsuite.c
       <test_name>_ns_tests.h
   secure
       CMakeLists.txt
       <test_name>_s_interface_testsuite.c
       <test_name>_s_tests.h
```

Creating test configurations

A test configuration controls whether one or multiple test suites are enabled. The doc TF-M Build Instructions. shows some test configurations which are already supported in current TF-M.

Developers shall assign corresponding test configurations to control the test suites.

Naming test configurations

The test configurations of example test suites are TEST_NS_<TEST_NAME> in non-secure and TEST_S_<TEST_NAME> in secure.

Note: The test configurations must be named with the prefixes TEST_S_ and TEST_NS_, for secure regression tests and non-secure regression tests respectively. Otherwise, tests_reg/test build system may not recognize it.

Setting test configurations

When the test configurations have dependencies, the default value need to be set. The setting is performed in following four steps.

1. Command line input: The configuration can be enabled or disabled by the command -DTEST_NS_<TEST_NAME>=ON/OFF -DTEST_S_<TEST_NAME>=ON/OFF, and the value cannot be changed throughout the whole compilation of TF-M tests.
2. tests_reg/test/config/config.cmake: The test configurations shall be OFF if its dependencies are not supported. The dependencies are probably from:
    1. TF-M partitions configurations like TFM_PARTITION_CRYPTO, TFM_PARTITION_INITIAL_ATTESTATION, etc.
    2. TF-M build mode configuration like CONFIG_TFM_SPM_BACKEND.
    3. TF-M other configurations like TFM_PARTITION_FIRMWARE_UPDATE, etc.
3. tests_reg/test/config/default_ns_test_config.cmake or tests_reg/test/config/default_s_test_config.cmake: It is required to give the default value of the new test configuration in these two files when TEST_NS or TEST_S is ON. The recommended value is ON unless the single test's code or data size is very large.
4. tests_reg/test/config/default_test_config.cmake: It is required to give the default value of the new test configuration in the file when both TEST_NS and TEST_S are OFF. The default value must be OFF.

Note: The test configurations must be set as CACHE value in CMAKE files. The CACHE set cannot replace the value from command line, see Set Cache Entry.

Checking test configurations

The new test configurations must be checked by function tfm_invalid_config() if they have any dependence. The value comes from command line may be wrong when the dependencies are conflicting.

Implement necessary checks in tests_reg/test/config/check_config.cmake.

Export TF-M configurations

If the new test depends on some TF-M configurations, export their value via tests_reg/test/config/config_ns_tests.cmake.in. TF-M secure build will install config_ns_tests.cmake and export configuration values. tf-m-tests non-secure build will include config_ns_tests.cmake and receive TF-M configuration values.

Applying test configurations

The mission of test configurations is to control the build. They are applied in test/secure_fw/suites/<test_name>/CMakeLists.txt like the example below.

```
if (NOT TEST_NS_<TEST_NAME> AND NOT TEST_S_<TEST_NAME>)
    return()
endif()

######################### Non Secure ################################
↪############

if (TEST_NS_<TEST_NAME>)
    add_library(tfm_test_suite_<test_name>_ns STATIC EXCLUDE_FROM_ALL)
    # target_sources()
    # target_include_directories()
    target_compile_definitions(tfm_test_suite_<test_name>_ns
```

(continues on next page)

(continued from previous page)

```
        INTERFACE
            TEST_NS_<TEST_NAME>
    )
    # target_link_libraries()
endif()

######################### Secure ################################
↪##############

if (TEST_S_<TEST_NAME>)
    add_library(tfm_test_suite_<test_name>_s STATIC EXCLUDE_FROM_ALL)
    # target_sources()
    # target_include_directories()
    target_compile_definitions(tfm_test_suite_<test_name>_s
        INTERFACE
            TEST_S_<TEST_NAME>
    )
    # target_link_libraries()
endif()
```

The function target_compile_definitions will export the macros TEST_NS_<TEST_NAME> or TEST_S_<TEST_NAME> into source code. and in the file tests_reg/test/framework/non_secure_suites.c or tests/framework/secure_suites.c, the definitions of these macros will be checked, and then the head file will be included and test cases will be registered if the macro is defined.

```
#ifdef TEST_NS_<TEST_NAME>
#include "<test_name>_ns_tests.h"
#endif

static struct test_suite_t test_suites[] = {
/* Non-secure example test cases */
    // ......
#ifdef TEST_NS_<TEST_NAME>
    {&register_testsuite_ns_<test_name>_interface, 0, 0, 0},
#endif
};
```

```
#ifdef TEST_S_<TEST_NAME>
#include "<test_name>_s_tests.h"
#endif

static struct test_suite_t test_suites[] = {
/* Secure example test cases */
    // ......
#ifdef TEST_S_<TEST_NAME>
    {&register_testsuite_s_<test_name>_interface, 0, 0, 0},
#endif
};
```

Note: On most platforms non-secure tests and secure tests run on the same CPU core, but dual-core platform is an exception. So secure test library and secure services shall be linked together in the file tests_reg/test/test/secure_fw/secure_tests.cmake. Thus they can be built on secure CPU core and non-secure tests library and RTOS are built on non-secure CPU core.

```
# ...
if (TEST_S_<TEST_NAME>)
    add_library(tfm_test_suite_<test_name>_s STATIC EXCLUDE_FROM_ALL)
```

(continues on next page)

(continued from previous page)

```
endif()
```

Adding a new test case in test suite

The test cases usually express as a function in source code. They will be added into an array with structure type called test_t defined in tests_reg/test/test/framework/test_framework.h.

```
struct test_t {
    TEST_FUN * const test;          /*!< Test function to call */
    const char *name;               /*!< Test name */
    const char *desc;               /*!< Test description */
};
```

For example, a new test case called TFM_NS_<TEST_NAME>_TEST_1001 is created and the function tfm_<test_name>_test_1001 needs to be defined in file <test_name>_ns_interface_testsuite.c. Then the function shall be appended into the array which will be quoted in function register_testsuite_ns_<test_name>_interface. See the reference code below.

```
/* List of test cases */
static void tfm_<test_name>_test_1001(struct test_result_t *ret);

/* Append test cases */
static struct test_t <test_name>_tests[] = {
    {&tfm_<test_name>_test_1001, "TFM_NS_<TEST_NAME>_TEST_1001",
    "Example test case"},
};

/* Register test case into test suites */
void register_testsuite_ns_<test_name>_interface(struct test_suite_t *p_test_suite)
{
    uint32_t list_size;

    list_size = (sizeof(<test_name>_tests) / sizeof(<test_name>_tests[0]));

    set_testsuite("<TEST_NAME> non-secure interface test (TFM_NS_<TEST_NAME>_TEST_1XXX)",
                  <test_name>_tests, list_size, p_test_suite);
}

static void tfm_<test_name>_test_1001(struct test_result_t *ret)
{
   /* test case code */
}
```

Adding test services

Some test group may need specific test services. These test services may support one or more groups thus developers shall determine the proper test scope. Refer to *Adding partitions for regression tests* to get more information.

Out-of-tree regression test suites

TF-M supports out-of-tree regression test suites build, whose source code folder is outside tf-m-tests repo. There are two configurations for developers to include the source code.

- EXTRA_NS_TEST_SUITE_PATH

  An absolute directory of the out-of-tree non-secure test suite source code folder. TF-M build system searches CMakeLists.txt of non-secure test suite in the source code folder.

- EXTRA_S_TEST_SUITE_PATH

  An absolute directory of the out-of-tree secure test suite source code folder.

Example usage

Take non-secure test as an example in tf-m-extras. A single out-of-tree test suite folder can be organized as the figure below:

```
extra_ns
  CMakeLists.txt
  ns_test.c
  ns_test_config.cmake
```

In the example above, EXTRA_NS_TEST_SUITE_PATH in the build command can be specified as listed below.

```
-DEXTRA_NS_TEST_SUITE_PATH=<Absolute-path-extra-test-folder>
```

Coding instructions

This is a demo of source code so the structure has been simplified. Files like .c and .h can be expanded to src and include folders respectively. The CMakeLists.txt is required in the root path and ns_test_config.cmake is optional.

Header Files

The header file extra_ns_tests.h must be included by out-of-tree source code. This file contains the declaration of void register_testsuite_extra_ns_interface(struct test_suite_t *p_test_suite).

Source code

To connect the out-of-tree source code and regression test framework, the test case function/functions must be defined first. An example format is:

```
void ns_test(struct test_result_t *ret)
{
    /* Add platform specific non-secure test suites code here. */

    ret->val = TEST_PASSED;
}
```

This function follows the standard TF-M test case function prototype.

Note: Extra tests can have one or more test cases. This is simplified example so only one test case is added.

After ns_test() is defined, a structure variable need to be created like:

```
static struct test_t plat_ns_t[] = {
    {&ns_test, "TFM_EXTRA_TEST_1001",
     "Extra Non-Secure test"},
};
```

It will be used by function register_testsuite_extra_ns_interface() to register the test by calling the set_testsuite() function:

```
void register_testsuite_extra_ns_interface(struct test_suite_t *p_test_suite)
{
    uint32_t list_size;

    list_size = (sizeof(plat_ns_t) /
                 sizeof(plat_ns_t[0]));

    set_testsuite("Extra Non-Secure interface tests"
                  "(TFM_NS_EXTRA_TEST_1XXX)",
                  plat_ns_t, list_size, p_test_suite);
}
```

The platform initialization code can be added in this function because it runs before ns_test().

Note: Function register_testsuite_extra_ns_interface() is declared in tf-m-tests repository without definition. It is supplied to out-of-tree source code and need to be defined with no change of its format, like returns error code and parameter name.

CMakeLists.txt

After extra test suite file were created they must be linked to tfm_test_suite_extra_ns CMAKE target:

```
target_sources(tfm_test_suite_extra_ns
    PRIVATE
        ${CMAKE_CURRENT_SOURCE_DIR}/ns_test.c
)
```

ns_test_config.cmake

The CMAKE configuration file is optional. If out-of-tree source already exists another configuration file, a new one can be ignored.

Copyright (c) 2021-2022, Arm Limited. All rights reserved. Copyright (c) 2022 Cypress Semiconductor Corporation (an Infineon company) or an affiliate of Cypress Semiconductor Corporation. All rights reserved.

Copyright (c) 2023, Arm Limited. All rights reserved.

Psa-arch-tests Documents PSA Architecture Tests

PSA APIs Architecture Test Suite

Introduction

Arm *Platform Security Architecture* (PSA) is a holistic set of threat models, security analyses, hardware and firmware architecture specifications, and an open source firmware reference implementation. PSA provides a recipe, based on industry best practice, that allows security to be consistently designed in, at both a hardware and firmware level. One of the PSA goals is to make IoT security easier and quicker for everyone. This means having reliable, consistent APIs and useful built-in security functions for device manufacturers and the developer community. These PSA APIs provides a consistent developer experience, hiding the underlying complexity of the security system.

For more information, visit the PSA Webpage.

Architecture test suite

The current implementation of the architecture test suite contains tests for PSA Firmware Framework and PSA Certified APIs specifications. The tests are available as open source. The tests and the corresponding abstraction layers are available with an Apache v2.0 license, allowing external contribution.

PSA Firmware Framework (PSA-FF)

The test suite for this specification is located in the **ff** directory of this repository. See the *PSA Firmware Framework README* file for more details.

PSA Certified APIs

The test suite for this specification is located in the **dev_apis** directory of this repository. See the *PSA Certified APIs README* file for more details.

License

Arm PSA test suite is distributed under Apache v2.0 License.

Feedback, contributions, and support

- For feedback, use the GitHub Issue Tracker that is associated with this repository.
- For support, send an email to support-psa-arch-tests@arm.com with details.
- Arm licensees can contact Arm directly through their partner managers.
- Arm welcomes code contributions through GitHub pull requests.

Copyright (c) 2018-2022, Arm Limited and Contributors. All rights reserved.

3.5 Motor Control

3.5.1 FreeMASTER

Communication Driver User Guide

Introduction

What is FreeMASTER? FreeMASTER is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.
- **USB** direct connection to target microcontroller
- **CAN bus**
- **TCP/IP network** wired or WiFi

- **Segger J-Link RTT**
- **JTAG** debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called "packet-driven BDM" interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to "packet-driven BDM", the FreeMASTER also supports a communication over [J-Link RTT]((https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

Driver version 3 This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to FreeMASTER community or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

Note: Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

Target platforms The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the src/platforms directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.
- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called FMSTR_TRANSPORT with a

uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.

- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The *mcuxsdk* folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The "ampsdk" drivers target automotive-specific MCUs and their respective SDKs. The "dreg" implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.

  The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

Replacing existing drivers For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

Clocks, pins, and peripheral initialization The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the FMSTR_Init function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

MCUXpresso SDK The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a "middleware" component which may be downloaded along with the example applications from https://mcuxpresso.nxp.com/en/welcome.

MCUXpresso SDK on GitHub

The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- The official FreeMASTER middleware repository.
- Online version of this document

FreeMASTER in Zephyr

The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

Example applications MCUX SDK Example applications

There are several example applications available for each supported MCU platform.

- **fmstr_uart** demonstrates a plain serial transmission, typically connecting to a computer's physical or virtual COM port. The typical transmission speed is 115200 bps.
- **fmstr_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr_usb_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer's USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr_wifi** is the fmstr_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr_net and fmstr_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.
- **fmstr_pdbdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM, the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.
- **fmstr_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER "middleware" driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

Zephyr sample spplications

Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

Features The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.
- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.
- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

Board Detection

The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.
- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

Memory Read

This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

Memory Write

Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

Masked Memory Write

To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

Oscilloscope

The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

Recorder

The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

TSA

With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

TSA Safety

When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

Application commands

The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

Pipes

The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

Serial single-wire operation

The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- "External" single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.
- "True" single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR_SERIAL_SINGLEWIRE configuration option.

Multi-session support

With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

Zephyr-specific

Dedicated communication task

FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR_Init and FMSTR_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

Zephyr shell and logging over FreeMASTER pipe

FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

Automatic TSA tables

TSA tables can be declared as "automatic" in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

Driver files The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- ***src/platforms*** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- ***src/common*** folder—contains the common driver source files shared by the driver for all supported platforms. All the *.c* files must be added to the project, compiled, and linked together with the application.
  - *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
  - *freemaster_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.

- *freemaster_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster_cfg.h* file.
- *freemaster_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
- *freemaster_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.
- *freemaster_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
- *freemaster_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
- *freemaster_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
- *freemaster_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
- *freemaster_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
- *freemaster_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
- *freemaster_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster_cfg.h* file.
- *freemaster_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
- *freemaster_serial.h* - defines the low-level character-oriented Serial API.
- *freemaster_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
- *freemaster_can.h* - defines the low-level message-oriented CAN API.
- *freemaster_net.c* - implements the Network protocol transport logic including multiple session management code.
- *freemaster_net.h* - definitions related to the Network transport.
- *freemaster_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
- *freemaster_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
- *freemaster_utils.h* - definitions related to utility code.

- ***src/drivers/[sdk]/serial*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_serial_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.
- ***src/drivers/[sdk]/can*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- ***src/drivers/[sdk]/network*** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
  - *freemaster_net_lwip_tcp.c* and *_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
  - *freemaster_net_segger_rtt.c* - implementation of network transport using Segger J-Link RTT interface

Driver configuration The driver is configured using a single header file (*freemaster_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster_cfg.h.example* file, rename it to *freemaster_cfg.h*, and save it into the project area.

Note: It is NOT recommended to leave the *freemaster_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

Configurable items This section describes the configuration options which can be defined in *freemaster_cfg.h*.

Interrupt modes

```
#define FMSTR_LONG_INTR   [0|1]
#define FMSTR_SHORT_INTR  [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

Value Type

boolean (0 or 1)

Description

Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See *Driver interrupt modes*.

- FMSTR_LONG_INTR — long interrupt mode
- FMSTR_SHORT_INTR — short interrupt mode
- FMSTR_POLL_DRIVEN — poll-driven mode

Note: Some options may not be supported by all communication interfaces. For example, the FMSTR_SHORT_INTR option is not supported by the USB_CDC interface.

Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

Value Type

Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

Description

Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- **FMSTR_SERIAL** - serial communication protocol
- **FMSTR_CAN** - using CAN communication
- **FMSTR_PDBDM** - using packet-driven BDM communication
- **FMSTR_NET** - network communication using TCP or UDP protocol

Serial transport

This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

FMSTR_SERIAL_DRV

Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

Value Type

Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

Description

When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- **FMSTR_SERIAL_MCUX_UART** - UART driver
- **FMSTR_SERIAL_MCUX_LPUART** - LPUART driver
- **FMSTR_SERIAL_MCUX_USART** - USART driver
- **FMSTR_SERIAL_MCUX_MINIUSART** - miniUSART driver
- **FMSTR_SERIAL_MCUX_QSCI** - DSC QSCI driver
- **FMSTR_SERIAL_MCUX_USB** - USB/CDC class driver (also see code in the */support/mcuxsdk_usb* folder)
- **FMSTR_SERIAL_56F800E_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as FMSTR_SERIAL_DRV. For example:

- **FMSTR_SERIAL_DREG_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

FMSTR_SERIAL_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

Value Type

Optional address value (numeric or symbolic)

Description

Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetSerialBaseAddress() to select the peripheral module.

FMSTR_COMM_BUFFER_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

Value Type

0 or a value in range 32...255

Description

Specify the size of the communication buffer to be allocated by the driver.
Default value, which suits all driver features, is used when this option is defined as 0.

FMSTR_COMM_RQUEUE_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

Value Type

Value in range 0...255

Description

Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR_SHORT_INTR interrupt mode.
The default value is 32 B.

FMSTR_SERIAL_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

Value Type

Boolean 0 or 1.

Description

Set to non-zero to enable the “True” single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

CAN Bus transport

This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

FMSTR_CAN_DRV

Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

Value Type

Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

Description

When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR_CAN_MCUX_FLEXCAN** - FlexCAN driver
- **FMSTR_CAN_MCUX_MCAN** - MCAN driver
- **FMSTR_CAN_MCUX_MSCAN** - msCAN driver
- **FMSTR_CAN_MCUX_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR_CAN_MCUX_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR_CAN_DRV.

FMSTR_CAN_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

Value Type

Optional address value (numeric or symbolic)

Description

Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetCanBaseAddress() to select the peripheral module.

FMSTR_CAN_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

Value Type

CAN identifier (11-bit or 29-bit number)

Description

CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Default value is 0x7AA.

FMSTR_CAN_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

Value Type

CAN identifier (11-bit or 29-bit number)

Description

CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

FMSTR_FLEXCAN_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

Value Type

Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description

Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

FMSTR_FLEXCAN_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

Value Type

Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description

Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

Network transport

This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

FMSTR_NET_DRV

Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

Value Type

Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

Description

When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR_NET_LWIP_TCP** - TCP communication using lwIP stack
- **FMSTR_NET_LWIP_UDP** - UDP communication using lwIP stack
- **FMSTR_NET_SEGGER_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR_CAN_DRV.

Add another row below:

FMSTR_NET_PORT

```
#define FMSTR_NET_PORT [number]
```

Value Type

TCP or UDP port number (short integer)

Description

Specifies the server port number used by TCP or UDP protocols.

FMSTR_NET_BLOCKING_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

Value Type

Timeout as number of milliseconds

Description

This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

FMSTR_NET_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

Value Type

Boolean 0 or 1.

Description

This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

Debugging options

FMSTR_DISABLE

```
#define FMSTR_DISABLE [0|1]
```

Value Type

boolean (0 or 1)

Description

Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

FMSTR_DEBUG_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

Value Type

Boolean 0 or 1.

Description

Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR_Poll() function to be called periodically. Default value is 0 (false).

FMSTR_APPLICATION_STR

```
#define FMSTR_APPLICATION_STR
```

Value Type

String.

Description

Name of the application visible in FreeMASTER host application.

Memory access

FMSTR_USE_READMEM

```
#define FMSTR_USE_READMEM [0|1]
```

Value Type

Boolean 0 or 1.

Description

Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature. Default value is 1 (true).

FMSTR_USE_WRITEMEM

```
#define FMSTR_USE_WRITEMEM [0|1]
```

Value Type

Boolean 0 or 1.

Description

Define as non-zero to implement the Memory Write command.
The default value is 1 (true).

Oscilloscope options

FMSTR_USE_SCOPE

```
#define FMSTR_USE_SCOPE [number]
```

Value Type

Integer number.

Description

Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature. Default value is 0.

FMSTR_MAX_SCOPE_VARS

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

Value Type

Integer number larger than 2.

Description

Number of variables to be supported by each Oscilloscope instance.
Default value is 8.

Recorder options

FMSTR_USE_RECORDER

```
#define FMSTR_USE_RECORDER [number]
```

Value Type

Integer number.

Description

Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.
Default value is 0.

FMSTR_REC_BUFF_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

Value Type

Integer number larger than 2.

Description

Defines the size of the memory buffer used by the Recorder instance #0.
Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

Value Type

Number (nanoseconds time).

Description

Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR_REC_BASE_SECONDS(x)
- FMSTR_REC_BASE_MILLISEC(x)
- FMSTR_REC_BASE_MICROSEC(x)
- FMSTR_REC_BASE_NANOSEC(x)

Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_FLOAT_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

Value Type

Boolean 0 or 1.

Description

Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.

Default value is 0 (false).

Application Commands options

FMSTR_USE_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

Value Type

Boolean 0 or 1.

Description

Define as non-zero to implement the Application Commands feature.
Default value is 0 (false).

FMSTR_APPCMD_BUFF_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

Value Type

Numeric buffer size in range 1..255

Description

The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

FMSTR_MAX_APPCMD_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

Value Type

Number in range 0..255

Description

The number of different Application Commands that can be assigned a callback handler function using FMSTR_RegisterAppCmdCall(). Default value is 0.

TSA options

FMSTR_USE_TSA

```
#define FMSTR_USE_TSA [0|1]
```

Value Type

Boolean 0 or 1.

Description

Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool.
Default value is 0 (false).

FMSTR_USE_TSA_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

Value Type

Boolean 0 or 1.

Description

Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables.
Default value is 0 (false).

FMSTR_USE_TSA_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

Value Type

Boolean 0 or 1.

Description

Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project.
Default value is 0 (false).

FMSTR_USE_TSA_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

Value Type

Boolean 0 or 1.

Description

Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR_SetUpTsaBuff() and FMSTR_TsaAddVar() functions.
Default value is 0 (false).

Pipes options

FMSTR_USE_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

Value Type

Boolean 0 or 1.

Description

Enable the FreeMASTER Pipes feature to be used.
Default value is 0 (false).

FMSTR_MAX_PIPES_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

Value Type

Number in range 1..63.

Description

The number of simultaneous pipe connections to support.
The default value is 1.

Driver interrupt modes To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

Completely Interrupt-Driven operation

Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from that handler.

Mixed Interrupt and Polling Modes

Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR_SerialIsr, FMSTR_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR_Poll* routine. Call *FMSTR_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed.

Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR_COMM_RQUEUE_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR_Poll* routine. No interrupts are needed and the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR_Poll* function is called by the application at least once per the serial "character time" which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR_SHORT_INTR* and *FMSTR_POLL_DRIVEN*), the protocol handling takes place in the FMSTR_Poll routine. An application interrupt can occur in the middle of the Read Memory or Write Memory commands' execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (FMSTR_LONG_INTR), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

Data types Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the FMSTR_ prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the fmstr_ prefix.

Communication interface initialization The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the FMSTR_Init call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the FMSTR_SerialIsr function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver.

For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the FMSTR_CanIsr function from the application handler.

Note: It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

FreeMASTER Recorder calls When using the FreeMASTER Recorder in the application (FMSTR_USE_RECORDER > 0), call the FMSTR_RecorderCreate function early after FMSTR_Init to set up each recorder instance to be used in the application. Then call the FMSTR_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR_Recorder in the main application loop.

In applications where FMSTR_Recorder is called periodically with a constant period, specify the period in the Recorder configuration structure before calling FMSTR_RecorderCreate. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

Driver usage Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all **.c* files of the FreeMASTER driver from the *src/common/platforms/[your_platform]* folder are a part of the project. See *Driver files* for more details.
- Configure the FreeMASTER driver by creating or editing the *freemaster_cfg.h* file and by saving it into the application project directory. See *Driver configuration* for more details.
- Include the *freemaster.h* file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the FMSTR_LONG_INTR and FMSTR_SHORT_INTR modes, install the application-specific interrupt routine and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from this handler.
- Call the FMSTR_Init function early on in the application initialization code.
- Call the FMSTR_RecorderCreate functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the FMSTR_Poll API function periodically when the application is idle.
- For the FMSTR_SHORT_INTR and FMSTR_LONG_INTR modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

Communication troubleshooting The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the FMSTR_DEBUG_TX option in the *freemaster_cfg.h* file and call the FMSTR_Poll function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

Control API There are three key functions to initialize and use the driver.

FMSTR_Init

Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description

This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

FMSTR_Poll

Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description

In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR_Poll* function is called during the "idle" time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the FMSTR_Poll function is called at least once per the time calculated as:

*N * Tchar*

where:

- *N* is equal to the length of the receive FIFO queue (configured by the FMSTR_COMM_RQUEUE_SIZE macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

Note: In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster_cfg.h* file.

FMSTR_SerialIsr / FMSTR_CanIsr

Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

Description

This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see *Driver interrupt modes*), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

Note: In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

Recorder API FMSTR_RecorderCreate

Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description

This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR_USE_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance *0* which may be automatically configured by FMSTR_Init when the *freemaster_cfg.h* configuration file defines the *FMSTR_REC_BUFF_SIZE* and *FMSTR_REC_TIMEBASE* options.

For more information, see *Configurable items*.

FMSTR_Recorder

Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description

This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

FMSTR_RecorderTrigger

Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description

This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

Fast Recorder API The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

TSA Tables When the TSA is enabled in the FreeMASTER driver configuration file (by setting the FMSTR_USE_TSA macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

TSA table definition

The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR_TSA_TABLE_BEGIN macro with a *table_id* identifying the table. The *table_id* shall be a valid C-langiage symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type)  /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type)  /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
FMSTR_TSA_MEMBER(struct_name, member_name, type)  /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR_TSA_TABLE_END macro:

```
FMSTR_TSA_TABLE_END()
```

TSA descriptor parameters

The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.
- *member_name* — structure member name.

Note: The structure member descriptors (FMSTR_TSA_MEMBER) must immediately follow the parent structure descriptor (FMSTR_TSA_STRUCT) in the table.

Note: To write-protect the variables in the FreeMASTER driver (FMSTR_TSA_RO_VAR), enable the TSA-Safety feature in the configuration file.

TSA variable types

The table lists *type* identifiers which can be used in TSA descriptors:

Constant	Description
FMSTR_TSA_UINT*n*	Unsigned integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_SINT*n*	Signed integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_FRAC*n*	Fractional number of size *n* bits (n=16,32,64).
FMSTR_TSA_FRAC_Q(*m*,*n*)	Signed fractional number in general Q form (m+n+1 total bits)
FMSTR_TSA_FRAC_UQ(*m*,*n*)	Unsigned fractional number in general UQ form (m+n total bits)
FMSTR_TSA_FLOAT	4-byte standard IEEE floating-point type
FMSTR_TSA_DOUBLE	8-byte standard IEEE floating-point type
FMSTR_TSA_POINTER	Generic pointer type defined (platform-specific 16 or 32 bit)
FMSTR_TSA_USERTYPE(*name*)	Structure or union type declared with FMSTR_TSA_STRUCT record

TSA table list

There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

TSA Active Content entries

FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files")     /* entering a new virtual directory */

/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index))          /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")

/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

TSA API FMSTR_SetUpTsaBuff

Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

Description

This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR_USE_TSA_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

FMSTR_TsaAddVar

Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR
↪tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object
- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
  - *FMSTR_TSA_INFO_RO_VAR* — read-only memory-mapped object (typically a variable)
  - *FMSTR_TSA_INFO_RW_VAR* — read/write memory-mapped object
  - *FMSTR_TSA_INFO_NON_VAR* — other entry, describing structure types, structure members, enumerations, and other types

Description

This function can be called only when the dynamic TSA table is enabled by the FMSTR_USE_TSA_DYNAMIC configuration option and when the FMSTR_SetUpTsaBuff function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See *TSA table definition* for more details about the TSA table entries.

Application Commands API FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Description

This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the FMSTR_APPCMDRESULT_NOCMD constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the FMSTR_AppCmdAck call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The FMSTR_GetAppCmd function does not report the commands for which a callback handler function exists. If the FMSTR_GetAppCmd function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns FMSTR_APPCMDRESULT_NOCMD.

FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

Description

This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see *FMSTR_GetAppCmd*).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR_AppCmdAck call, copy the data out to a private buffer.

FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

Description

This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR_GetAppCmd function is FMSTR_APPCMDRESULT_NOCMD.

FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR_APPCMD_BUFF_SIZE value.

Description

This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR_GetAppCmdData and the data buffer after FMSTR_AppCmdSetResponseData is called.

Note: The current version of FreeMASTER does not support the Application Command response data.

FMSTR_RegisterAppCmdCall

Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↪PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

Return value

This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR_MAX_APPCMD_CALLS different Application Commands.

Description

This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
    FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

Note: The FMSTR_MAX_APPCMD_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR_MAX_APPCMD_CALLS is undefined or defined as zero, the FMSTR_RegisterAppCmdCall function always fails.

Pipes API FMSTR_PipeOpen

Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
↪
    FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
    FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
    FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR_PIPE_MODE_xxx and FMSTR_PIPE_SIZE_xxx constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

Description

This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

FMSTR_PipeClose

Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call

Description

This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

FMSTR_PipeWrite

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

Description

This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk. This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the nGranularity value equal to the nLength value, all data are considered as one chunk which is either written successfully as a whole or not at all. The nGranularity value of 0 or 1 disables the data-chunk approach.

FMSTR_PipeRead

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

Description

This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The readGranularity argument can be used to copy the data in larger chunks in the same way as described in the FMSTR_PipeWrite function.

API data types This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

Note: The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

Public common types

The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

Type name	Description
FMSTR_ADDR	Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type.
For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations.	
FMSTR_SIZE	Data type used to hold the memory block size.
It is required that this type is unsigned and at least 16 bits wide integer.	
FMSTR_BOOL	Data type used as a general boolean type.
This type is used only in zero/non-zero conditions in the driver code.	
FMSTR_APPCM	Data type used to hold the Application Command code.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to create the Application Command data buffer.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to hold the Application Command result code.
Generally, this is an unsigned 8-bit value.	

Public TSA types

The table describes the TSA-specific public data types. These types are declared in the *freemaster_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

FMSTR_TSA_TII	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.
By default, this is defined as FMSTR_SIZE.	
FMSTR_TSA_TS	Data type used to hold a memory block size, as used in the TSA descriptors.
By default, this is defined as FMSTR_SIZE.	

Public Pipes types

The table describes the data types used by the FreeMASTER Pipes API:

FMSTR_HPIPE	Pipe handle that identifies the open-pipe object.
Generally, this is a pointer to a void type.	
FMSTR_PIPE_P(	Integer type required to hold at least 7 bits of data.
Generally, this is an unsigned 8-bit or 16-bit type.	
FMSTR_PIPE_SI	Integer type required to hold at least 16 bits of data.
This is used to store the data buffer sizes.	
FMSTR_PPIPEF	Pointer to the pipe handler function.
See *FMSTR_PipeOpen* for more details.	

Internal types

The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

FMSTR_U8	The smallest memory entity. On the vast majority of platforms, this is an unsigned 8-bit integer. On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.
FMSTR_U16	Unsigned 16-bit integer.
FMSTR_U32	Unsigned 32-bit integer.
FMSTR_S8	Signed 8-bit integer.
FMSTR_S16	Signed 16-bit integer.
FMSTR_S32	Signed 32-bit integer.
FMSTR_FLOAT	4-byte standard IEEE floating-point type.
FMSTR_FLAGS	Data type forming a union with a structure of flag bit-fields.
FMSTR_SIZE8	Data type holding a general size value, at least 8 bits wide.
FMSTR_INDEX	General for-loop index. Must be signed, at least 16 bits wide.
FMSTR_BCHR	A single character in the communication buffer. Typically, this is an 8-bit unsigned integer, except for the DSP platforms where it is a 16-bit integer.
FMSTR_BPTR	A pointer to the communication buffer (an array of FMSTR_BCHR).

Document references

Links

- This document online: https://mcuxpresso.nxp.com/mcuxsdk/latest/html/middleware/freemaster/doc/index.html

- FreeMASTER tool home: www.nxp.com/freemaster
- FreeMASTER community area: community.nxp.com/community/freemaster
- FreeMASTER GitHub code repo: https://github.com/nxp-mcuxpresso/mcux-freemaster
- MCUXpresso SDK home: www.nxp.com/mcuxpresso
- MCUXpresso SDK builder: mcuxpresso.nxp.com/en

Documents

- *FreeMASTER Usage Serial Driver Implementation* (document AN4752)
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document AN4771)
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document AN4860)

Revision history This Table summarizes the changes done to this document since the initial release.

Revision	Date	Description
1.0	03/2006	Limited initial release
2.0	09/2007	Updated for FreeMASTER version. New Freescale document template used.
2.1	12/2007	Added description of the new Fast Recorder feature and its API.
2.2	04/2010	Added support for MPC56xx platform, Added new API for use CAN interface.
2.3	04/2011	Added support for Kxx Kinetis platform and MQX operating system.
2.4	06/2011	Serial driver update, adds support for USB CDC interface.
2.5	08/2011	Added Packet Driven BDM interface.
2.7	12/2013	Added FLEXCAN32 interface, byte access and isr callback configuration option.
2.8	06/2014	Removed obsolete license text, see the software package content for up-to-date license.
2.9	03/2015	Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.
3.0	08/2016	Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.
4.0	04/2019	Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms.
4.1	04/2020	Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.
4.2	09/2020	Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.
4.3	10/2024	Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.
4.4	04/2025	Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.

3.6 Multicore SDK

Multicore Software Development Kit (MCSDK) provides comprehensive software support for NXP multicore devices. The MCSDK is combined with the MCUXpresso SDK to form a framework for easy development of multicore applications.

3.6.1 Quick links

- *Release Notes*

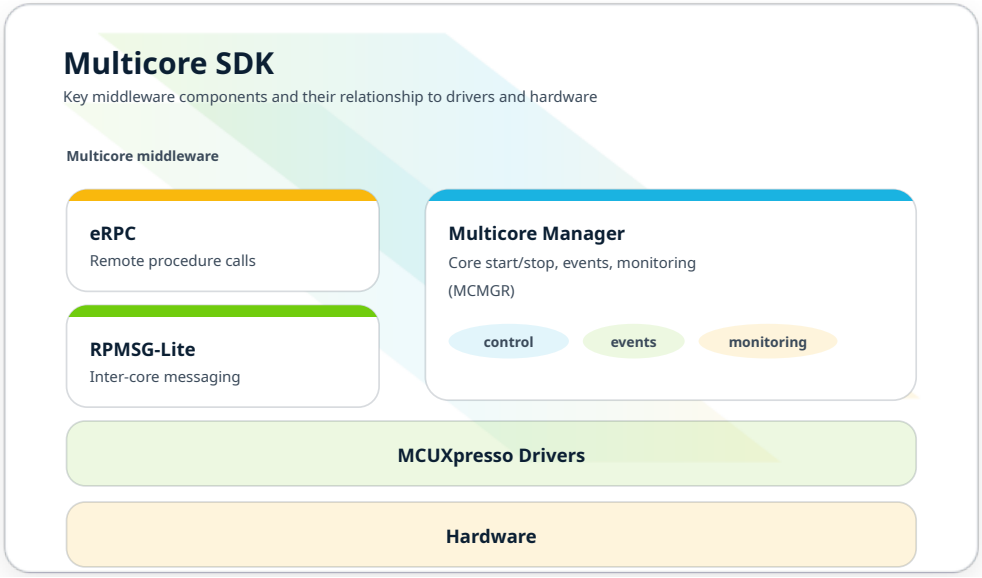


Fig. 19: Multicore SDK software stack (simplified).

- *Getting Started*
- *Multicore SDK Changelog*

3.6.2 Components

Component	Description
RPMSG-Lite	Lightweight implementation of the Remote Processor Messaging (RPMsg) protocol for inter-core messaging.
Multicore Manager	Services for multicore systems (core start/stop, events, and monitoring).
eRPC	Embedded RPC system suitable for multicore and multiprocessor designs.

MCSDK Release Notes

Multicore SDK (MCSDK) Release Notes Overview

These are the release notes for the NXP **Multicore Software Development Kit (MCSDK)**.

Item	Value
MCSDK version	**26.09.00**
Scope	Multicore middleware components and examples
Location in MCUXpresso SDK	<MCUXpressoSDK_install_dir>/middleware/multicore/

The MCSDK is combined with the MCUXpresso SDK to provide a framework for developing multicore applications on NXP devices.

What's new

Component updates

Review detailed changes per component:

- eRPC: *CHANGELOG*
- RPMSG-Lite: *CHANGELOG*
- Multicore Manager (MCMGR): *CHANGELOG*

Supported evaluation boards

Boards with enabled multicore example coverage in this release:

Multicore examples
LPCXpresso55S69FRDM-K32L3A6MIMXRT1170-EVKBMIMXRT1160-EVKMIMXRT1180-EVKMCX-N5XX-EVKMCX-N9XX-EVKFRDM-MCXN947MIMXRT700-EVKKW47-EVKKW47-LOCFRDM-MCXW72MCX-W72-EVKFRDM-IMXRT1186

Development tools

The MCSDK is compiled and tested with the toolchains described in the MCUXpresso SDK documentation:

- https://mcuxpresso.nxp.com/mcuxsdk/latest/html/gsd/installation.html#toolchain

Release contents

This table describes the release contents. Not all MCUXpresso SDK packages contain the full set of these components.

Deliverable	Location
Multicore SDK location (<MCSDK_dir>)	<MCUXpressoSDK_install_dir>/middleware/multicore/
Documentation	<MCSDK_dir>/mcuxsdk-doc/
Embedded Remote Procedure Call (eRPC)	<MCSDK_dir>/erpc/
Multicore Manager (MCMGR)	<MCSDK_dir>/mcmgr/
RPMSG-Lite	<MCSDK_dir>/rpmsg-lite/
Multicore demo applications	<MCUXpressoSDK_install_dir>/examples/multicore_examples/

MCSDK component overview

Together, the MCSDK and the MCUXpresso SDK form a framework for developing software for NXP multicore devices. The MCSDK release includes:

- Embedded Remote Procedure Call (eRPC)
- Multicore Manager (MCMGR)
- Remote Processor Messaging – Lite (RPMSG-Lite)

The MCSDK also includes documentation and demo applications for multicore communication.

Demo applications

The demo applications demonstrate MCSDK component usage on supported development boards.

Examples are located under:

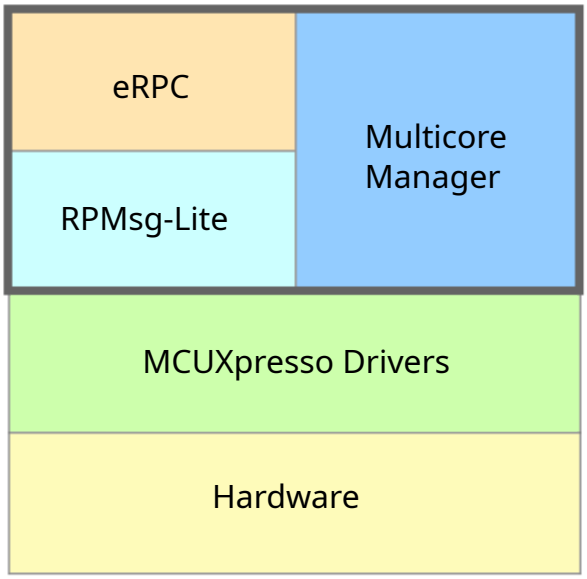
- Multicore: <MCUXpressoSDK_install_dir>/examples/multicore_examples/

Multicore examples
freertos_message_buffershello_worldmulticore_managerrpmsg_lite_pingpongrpmsg_lite_pingpong_rtosrpmsg_lite_

MCSDK Getting Started

Getting Started with Multicore SDK (MCSDK) Overview

The **Multicore Software Development Kit (MCSDK)** provides software components for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to provide a framework for developing multicore applications.



All MCSDK-related files are located in:

- <MCUXpressoSDK_install_dir>/middleware/multicore/

For supported toolchain versions, see the release notes.

Quick links

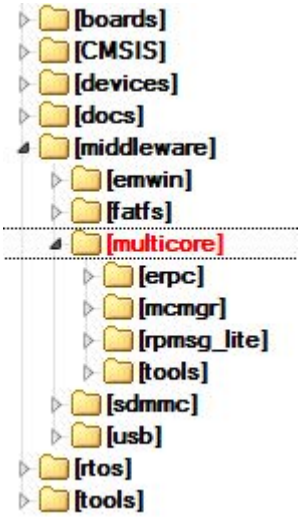
- MCSDK Release Notes
- *MCSDK Changelog*
- Component repositories (within MCSDK):
  - RPMSG-Lite
  - Multicore Manager (MCMGR)
  - eRPC
- eRPC detailed Getting Started guide:

– eRPC Getting Started

MCSDK components

The MCSDK consists of the following components:

- **Embedded Remote Procedure Call (eRPC):** Library + code generator implementing transparent function calls to services running on another core.
- **Multicore Manager (MCMGR):** Core management services (start/stop secondary cores, events, monitoring).
- **Remote Processor Messaging – Lite (RPMSG-Lite):** Lightweight inter-processor messaging library based on RPMsg.



Where things live

Item	Path
MCSDK root	<MCUXpressoSDK_install_dir>/middleware/multicore/
eRPC	<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/
MCMGR	<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/
RPMSG-Lite	<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg-lite/
MCSDK documentation	<MCUXpressoSDK_install_dir>/middleware/multicore/mcuxsdk-doc/
Multicore examples	<MCUXpressoSDK_install_dir>/examples/multicore_examples/

MCSDK demo applications

Multicore example applications are stored together with other MCUXpresso SDK examples.

Location	Folder
Multicore example projects	<MCUXpressoSDK_install_dir>/examples/multicore_examples/<application_name>/

Each example application contains a README describing setup, required images, and runtime behavior.

Inter-Processor Communication (IPC) levels

The MCSDK provides several mechanisms for Inter-Processor Communication (IPC). Pick the lightest one that matches your requirements.

IPC using low-level drivers

NXP multicore SoCs commonly provide dedicated IPC peripherals (Mailbox for some LPC parts, Messaging Unit (MU) for some Kinetis and i.MX parts).

This is the most lightweight approach: using the Mailbox/MU driver API functions, it is possible to pass a value from core to core via registers (scalar or pointer to shared memory) and trigger inter-core interrupts for notifications.

For details about individual driver API functions, see the MCUXpresso SDK API Reference Manual of the specific device.

Messaging mechanism (RPMSG-Lite)

On top of Mailbox/MU drivers, a messaging system can be implemented, allowing messages to be sent between endpoints created on each CPU.

RPMSG-Lite provides this capability and is the preferred MCUXpresso SDK messaging library. It implements ring buffers in shared memory for message exchange.

This costs additional code and data memory, but provides a unified messaging API and helps portability across devices/RTOSes.

Remote procedure calls (eRPC)

To facilitate IPC further, eRPC allows remote function invocation via a simple local function call.

Using different transport layers, it can communicate:

- between individual cores of multicore SoCs (commonly via RPMSG-Lite)
- between separate processors (SPI, UART, TCP/IP)

If the communicating peer is a Linux user-space application, the generated code can be in C/C++ or Python.

The IPC levels demonstrate the scalability of the MCSDK. Based on application needs, different IPC techniques can be used depending on complexity, speed, memory, and system design.

Changelog Multicore SDK

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

[26.09.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.3.0
  - RPMsg-Lite v5.5.0

[26.06.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0

  - Multicore Manager (MCMgr) v5.2.0
  - RPMsg-Lite v5.4.1

[26.03.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.1.0
  - RPMsg-Lite v5.4.0

[25.12.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.2
  - RPMsg-Lite v5.3.0

[25.09.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.1
  - RPMsg-Lite v5.2.1

[25.06.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.0
  - RPMsg-Lite v5.2.0

[25.03.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.7
  - RPMsg-Lite v5.1.4

[24.12.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.6
  - RPMsg-Lite v5.1.3

[2.16.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.5
  - RPMsg-Lite v5.1.2

[2.15.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.12.0
  - eRPC generator (erpcgen) v1.12.0
  - Multicore Manager (MCMgr) v4.1.5
  - RPMsg-Lite v5.1.1

[2.14.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.11.0
  - eRPC generator (erpcgen) v1.11.0
  - Multicore Manager (MCMgr) v4.1.4
  - RPMsg-Lite v5.1.0

[2.13.0_imxrt1180a0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.10.0
  - eRPC generator (erpcgen) v1.10.0
  - Multicore Manager (MCMgr) v4.1.3
  - RPMsg-Lite v5.0.0

[2.13.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.10.0
  - eRPC generator (erpcgen) v1.10.0
  - Multicore Manager (MCMgr) v4.1.3
  - RPMsg-Lite v5.0.0

[2.12.0_imx93]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.1
  - eRPC generator (erpcgen) v1.9.1
  - Multicore Manager (MCMgr) v4.1.2
  - RPMsg-Lite v4.0.1

[2.12.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.1
  - eRPC generator (erpcgen) v1.9.1
  - Multicore Manager (MCMgr) v4.1.2
  - RPMsg-Lite v4.0.0

[2.11.1]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.0
  - eRPC generator (erpcgen) v1.9.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.2.1

[2.11.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.0
  - eRPC generator (erpcgen) v1.9.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.2.0

[2.10.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.8.1
  - eRPC generator (erpcgen) v1.8.1
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.1.2

[2.9.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.8.0
  - eRPC generator (erpcgen) v1.8.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.1.1

[2.8.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.4
  - eRPC generator (erpcgen) v1.7.4
  - Multicore Manager (MCMgr) v4.1.0
  - RPMsg-Lite v3.1.0

[2.7.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.3
  - eRPC generator (erpcgen) v1.7.3
  - Multicore Manager (MCMgr) v4.1.0
  - RPMsg-Lite v3.0.0

[2.6.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.2
  - eRPC generator (erpcgen) v1.7.2
  - Multicore Manager (MCMgr) v4.0.3
  - RPMsg-Lite v2.2.0

[2.5.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.1
  - eRPC generator (erpcgen) v1.7.1
  - Multicore Manager (MCMgr) v4.0.2
  - RPMsg-Lite v2.0.2

[2.4.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.0
  - eRPC generator (erpcgen) v1.7.0
  - Multicore Manager (MCMgr) v4.0.1
  - RPMsg-Lite v2.0.1

[2.3.1]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.6.0
  - eRPC generator (erpcgen) v1.6.0
  - Multicore Manager (MCMgr) v4.0.0
  - RPMsg-Lite v1.2.0

[2.3.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.5.0
  - eRPC generator (erpcgen) v1.5.0
  - Multicore Manager (MCMgr) v3.0.0
  - RPMsg-Lite v1.2.0

[2.2.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.4.0
  - eRPC generator (erpcgen) v1.4.0
  - Multicore Manager (MCMgr) v2.0.1
  - RPMsg-Lite v1.1.0

[2.1.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.3.0
  - eRPC generator (erpcgen) v1.3.0

[2.0.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.2.0
  - eRPC generator (erpcgen) v1.2.0
  - Multicore Manager (MCMgr) v2.0.0
  - RPMsg-Lite v1.0.0

[1.1.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.1.0
  - Multicore Manager (MCMgr) v1.1.0
  - Open-AMP / RPMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev01

[1.0.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.0.0
  - Multicore Manager (MCMgr) v1.0.0
  - Open-AMP / RPMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev00

RPMSG-Lite

Quick links

- *Changelog*
- API documentation (Doxygen): https://nxp-mcuxpresso.github.io/rpmsg-lite/

This documentation describes the RPMsg-Lite component, which is a lightweight implementation of the Remote Processor Messaging (RPMsg) protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system.

Compared to the RPMsg implementation of the Open Asymmetric Multi Processing (OpenAMP) framework (https://github.com/OpenAMP/open-amp), the RPMsg-Lite offers a code size reduction, API simplification, and improved modularity. On smaller Cortex-M0+ based systems, it is recommended to use RPMsg-Lite.

The RPMsg-Lite is an open-source component developed by NXP Semiconductors and released under the BSD-compatible license.

For overview please read RPMSG-Lite VirtIO Overview.

For RPMSG-Lite Design Considerations please read RPMSG-Lite Design Considerations.

Motivation to create RPMsg-Lite

There are multiple reasons why RPMsg-Lite was developed. One reason is the need for the small footprint of the RPMsg protocol-compatible communication component, another reason is the simplification of extensive API of OpenAMP RPMsg implementation.

RPMsg protocol was not documented, and its only definition was given by the Linux Kernel and legacy OpenAMP implementations. This has changed with [1] which is a standardization protocol allowing multiple different implementations to coexist and still be mutually compatible.

Small MCU-based systems often do not implement dynamic memory allocation. The creation of static API in RPMsg-Lite enables another reduction of resource usage. Not only does the dynamic allocation adds another 5 KB of code size, but also communication is slower and less deterministic, which is a property introduced by dynamic memory. The following table shows some rough comparison data between the OpenAMP RPMsg implementation and new RPMsg-Lite implementation:

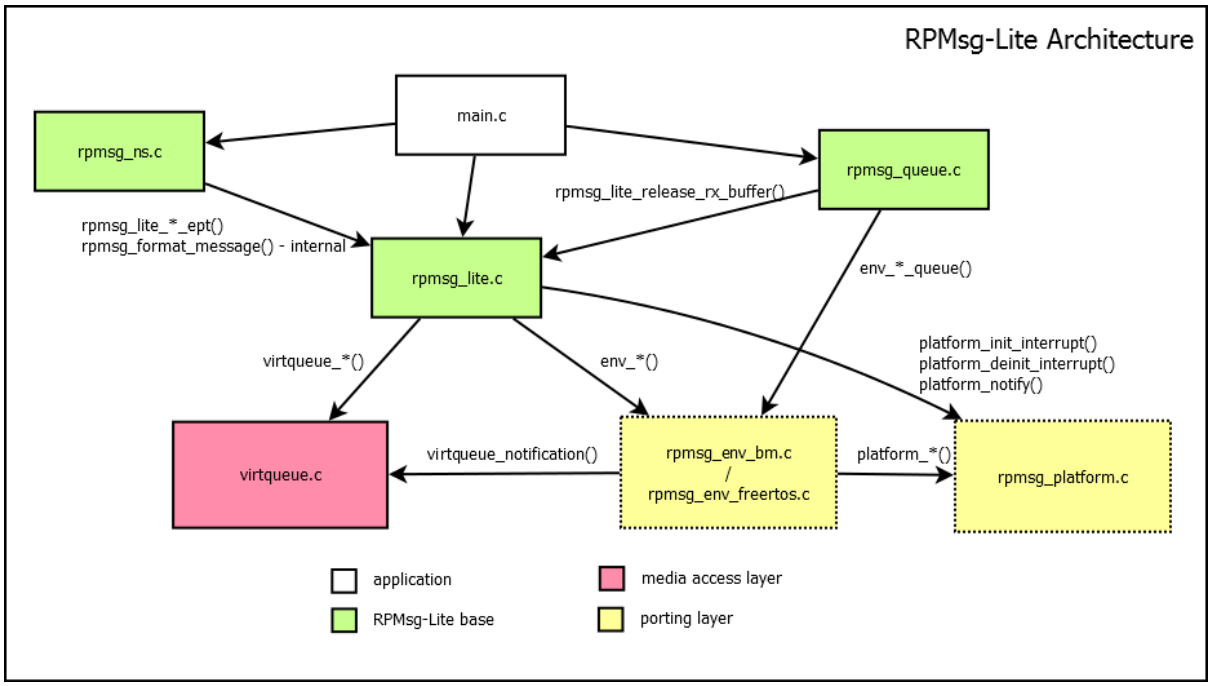
Component / Configuration	Flash [B]	RAM [B]
OpenAMP RPMsg / Release (reference)	5547	456 + dynamic
RPMsg-Lite / Dynamic API, Release	3462	56 + dynamic
Relative Difference [%]	~62.4%	~12.3%
RPMsg-Lite / Static API (no malloc), Release	2926	352
Relative Difference [%]	~52.7%	~77.2%

Implementation

The implementation of RPMsg-Lite can be divided into three sub-components, from which two are optional. The core component is situated in rpmsg_lite.c. Two optional components are used to implement a blocking receive API (in rpmsg_queue.c) and dynamic “named” endpoint creation and deletion announcement service (in rpmsg_ns.c).

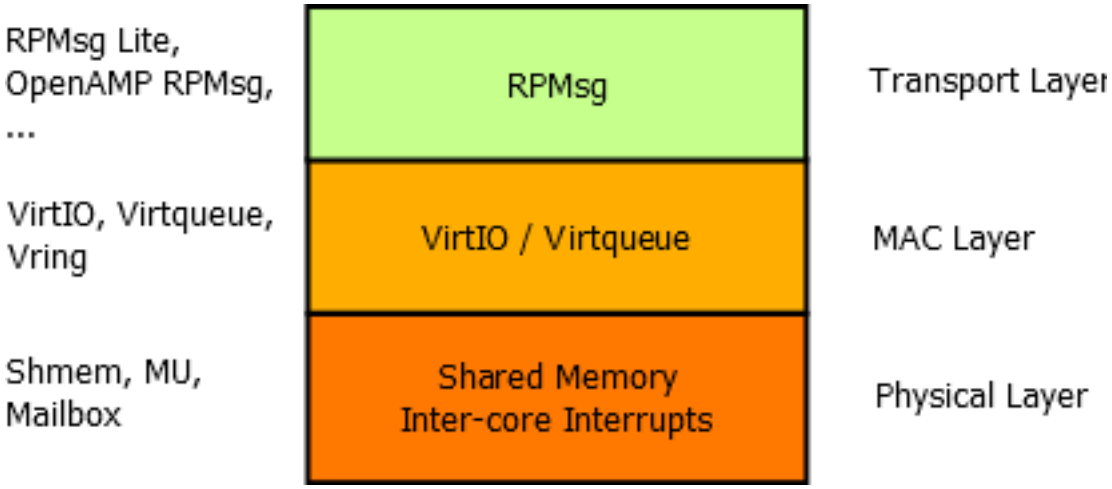
The actual “media access” layer is implemented in virtqueue.c, which is one of the few files shared with the OpenAMP implementation. This layer mainly defines the shared memory model, and internally defines used components such as vring or virtqueue.

The porting layer is split into two sub-layers: the environment layer and the platform layer. The first sublayer is to be implemented separately for each environment. (The bare metal environment already exists and is implemented in rpmsg_env_bm.c, and the FreeRTOS environment is implemented in rpmsg_env_freertos.c etc.) Only the source file, which matches the used environment, is included in the target application project. The second sublayer is implemented in rpmsg_platform.c and defines low-level functions for interrupt enabling, disabling, and triggering mainly. The situation is described in the following figure:



RPMsg-Lite core sub-component

This subcomponent implements a blocking send API and callback-based receive API. The RPMsg protocol is part of the transport layer. This is realized by using so-called endpoints. Each endpoint can be assigned a different receive callback function. However, it is important to notice that the callback is executed in an interrupt environment in current design. Therefore, certain actions like memory allocation are discouraged to execute in the callback. The following figure shows the role of RPMsg in an ISO/OSI-like layered model:



Queue sub-component (optional)

This subcomponent is optional and requires implementation of the env_*_queue() functions in the environment porting layer. It uses a blocking receive API, which is common in RTOS-environments. It supports both copy and nocopy blocking receive functions.

Name Service sub-component (optional)

This subcomponent is a minimum implementation of the name service which is present in the Linux Kernel implementation of RPMsg. It allows the communicating node both to send announcements about “named” endpoint (in other words, channel) creation or deletion and to receive these announcement taking any user-defined action in an application callback. The endpoint address used to receive name service announcements is arbitrarily fixed to be 53 (0x35).

Usage

The application should put the /rpmsg_lite/lib/include directory to the include path and in the application, include either the rpmsg_lite.h header file, or optionally also include the rpmsg_queue.h and/or rpmsg_ns.h files. Both porting sublayers should be provided for you by NXP, but if you plan to use your own RTOS, all you need to do is to implement your own environment layer (in other words, rpmsg_env_myrtos.c) and to include it in the project build.

The initialization of the stack is done by calling the rpmsg_lite_master_init() on the master side and the rpmsg_lite_remote_init() on the remote side. This initialization function must be called prior to any RPMsg-Lite API call. After the init, it is wise to create a communication endpoint, otherwise communication is not possible. This can be done by calling the rpmsg_lite_create_ept() function. It optionally accepts a last argument, where an internal context of the endpoint is created, just in case the RL_USE_STATIC_API option is set to 1. If not, the stack internally calls env_alloc() to allocate dynamic memory for it. In case a callback-based receiving is to be used, an ISR-callback is registered to each new endpoint with user-defined callback data pointer. If a blocking receive is desired (in case of RTOS environment), the rpmsg_queue_create() function must be called before calling rpmsg_lite_create_ept(). The queue handle is passed to the endpoint creation function as a callback data argument and the callback function is set to rpmsg_queue_rx_cb(). Then, it is possible to use rpmsg_queue_receive() function to listen on a queue object for incoming messages. The rpmsg_lite_send() function is used to send messages to the other side.

The RPMsg-Lite also implements no-copy mechanisms for both sending and receiving operations. These methods require specifics that have to be considered when used in an application.

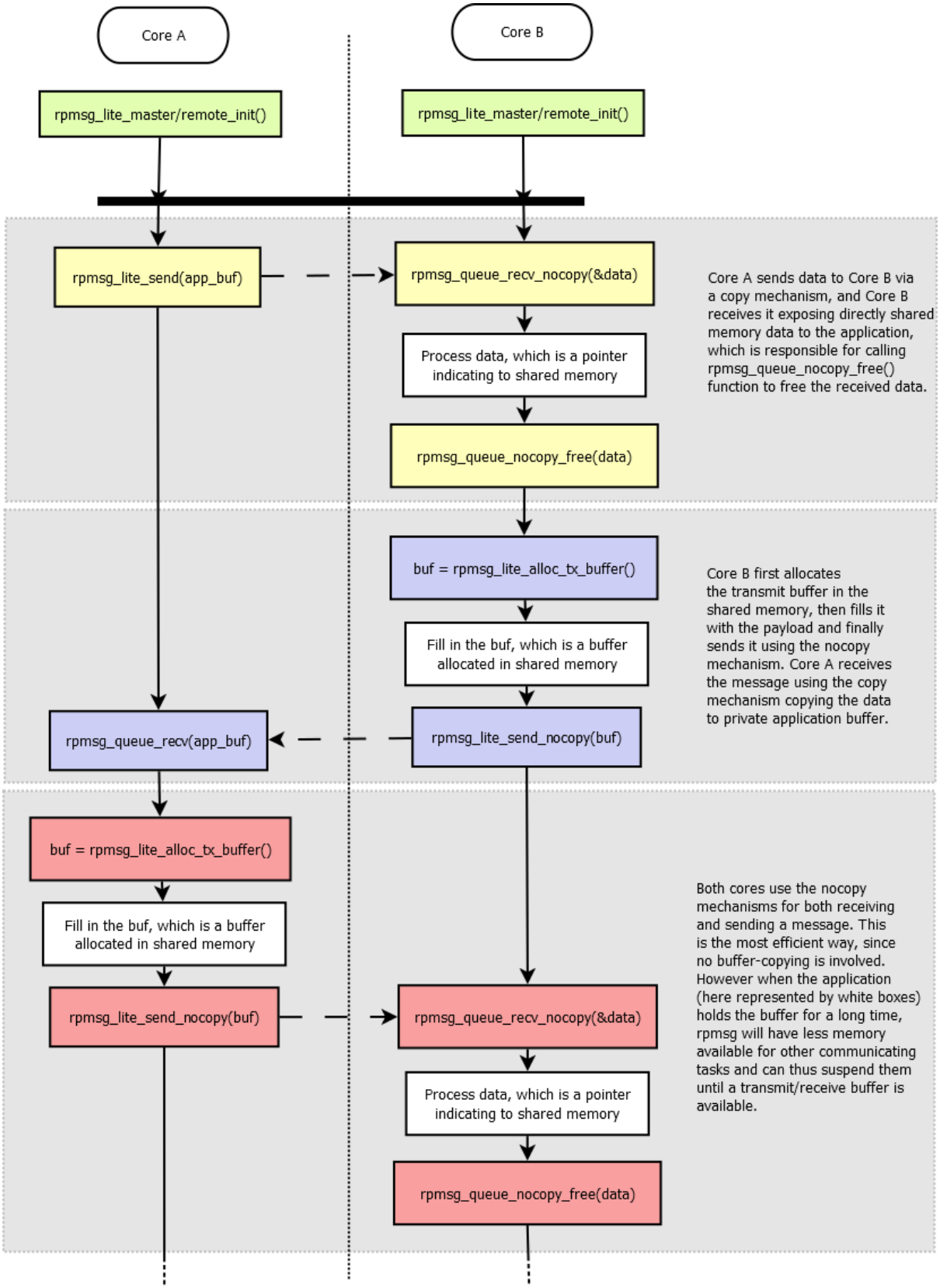
no-copy-send mechanism: This mechanism allows sending messages without the cost for copying data from the application buffer to the RPMsg/virtio buffer in the shared memory. The sequence of no-copy sending steps to be performed is as follows:

- Call the rpmsg_lite_alloc_tx_buffer() function to get the virtio buffer and provide the buffer pointer to the application.
- Fill the data to be sent into the pre-allocated virtio buffer. Ensure that the filled data does not exceed the buffer size (provided as the rpmsg_lite_alloc_tx_buffer() size output parameter).
- Call the rpmsg_lite_send_nocopy() function to send the message to the destination endpoint. Consider the cache functionality and the virtio buffer alignment. See the rpmsg_lite_send_nocopy() function description below.

no-copy-receive mechanism: This mechanism allows reading messages without the cost for copying data from the virtio buffer in the shared memory to the application buffer. The sequence of no-copy receiving steps to be performed is as follows:

- Call the rpmsg_queue_recv_nocopy() function to get the virtio buffer pointer to the received data.
- Read received data directly from the shared memory.
- Call the rpmsg_queue_nocopy_free() function to release the virtio buffer and to make it available for the next data transfer.

The user is responsible for destroying any RPMsg-Lite objects he has created in case of deinitialization. In order to do this, the function rpmsg_queue_destroy() is used to destroy a queue, rpmsg_lite_destroy_ept() is used to destroy an endpoint and finally, rpmsg_lite_deinit() is used to deinitialize the RPMsg-Lite intercore communication stack. Deinitialize all endpoints using a queue before deinitializing the queue. Otherwise, you are actively invalidating the used queue handle, which is not allowed. RPMsg-Lite does not check this internally, since its main aim is to be lightweight.



Examples

RPMsg_Lite multicore examples are part of NXP MCUXpressoSDK packages. Visit https://mcuxpresso.nxp.com to configure, build and download these packages. To get the board list with multicore support (RPMsg_Lite included) use filtering based on Middleware and search for

'multicore' string. Once the selected package with the multicore middleware is downloaded, see

<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples for RPMsg_Lite multicore examples with 'rpmsg_lite_' name prefix.

Another way of getting NXP MCUXpressoSDK RPMsg_Lite multicore examples is using the mcuxsdk-manifests Github repo. Follow the description how to use the West tool to clone and update the mcuxsdk-manifests repo in readme section. Once done the armgcc rpmsg_lite examples can be found in

mcuxsdk/examples/_<board_name>/multicore_examples

You can use the evkmimxrt1170 as the board_name for instance. Similar to MCUXpressoSDK packages the RPMsg_Lite examples use the 'rpmsg_lite_' name prefix.

Notes

Environment layers implementation

Several environment layers are provided in lib/rpmsg_lite/porting/environment folder. Not all of them are fully tested however. Here is the list of environment layers that passed testing:

- rpmsg_env_bm.c
- rpmsg_env_freertos.c
- rpmsg_env_xos.c
- rpmsg_env_threadx.c

The rest of environment layers has been created and used in some experimental projects, it has been running well at the time of creation but due to the lack of unit testing there is no guarantee it is still fully functional.

Shared memory configuration

It is important to correctly initialize/configure the shared memory for data exchange in the application. The shared memory must be accessible from both the master and the remote core and it needs to be configured as Non-Cacheable memory. Dedicated shared memory section in liker file is also a good practise, it is recommended to use linker files from MCUXpressSDK packages for NXP devices based applications. It needs to be ensured no other application part/component is unintentionally accessing this part of memory.

Configuration options

The RPMsg-Lite can be configured at the compile time. The default configuration is defined in the rpmsg_default_config.h header file. This configuration can be customized by the user by including rpmsg_config.h file with custom settings. The following table summarizes all possible RPMsg-Lite configuration options.

Configuration option	Default value	Usage
RL_MS_PE	(1)	Delay in milliseconds used in non-blocking API functions for polling.
RL_BUFFE	(496)	Size of the buffer payload, it must be more than 1 byte, and has to be word align (including rpmsg header size 16 bytes), if not it will be aligned up
RL_BUFFE	(2)	Number of the buffers, it must be power of two (2, 4, ...)
RL_API_H	(1)	Zero-copy API functions enabled/disabled.
RL_USE_S	(0)	Static API functions (no dynamic allocation) enabled/disabled.
RL_USE_D	(0)	Memory cache management of shared memory. Use in case of data cache is enabled for shared memory.
RL_CLEAF	(0)	Clearing used buffers before returning back to the pool of free buffers enabled/disabled.
RL_USE_M	(0)	When enabled IPC interrupts are managed by the Multicore Manager (IPC interrupts router), when disabled RPMsg-Lite manages IPC interrupts by itself.
RL_USE_E	(0)	When enabled the environment layer uses its own context. Required for some environments (QNX). The default value is 0 (no context, saves some RAM).
RL_DEBU(	(0)	When enabled buffer pointers passed to rpmsg_lite_send_nocopy() and rpmsg_lite_release_rx_buffer() functions (enabled by RL_API_HAS_ZEROCOPY config) are checked to avoid passing invalid buffer pointer. The default value is 0 (disabled). Do not use in RPMsg-Lite to Linux configuration.
RL_ALLO\	(0)	When enabled the opposite side is notified each time received buffers are consumed and put into the queue of available buffers. Enable this option in RPMsg-Lite to Linux configuration to allow unblocking of the Linux blocking send. The default value is 0 (RPMsg-Lite to RPMsg-Lite communication).
RL_ALLO\	(0)	It allows to define custom shared memory configuration and replacing the shared memory related global settings from rpmsg_config.h This is useful when multiple instances are running in parallel but different shared memory arrangement (vring size & alignment, buffers size & count) is required. The default value is 0 (all RPMsg_Lite instances use the same shared memory arrangement as defined by common config macros).
RL_ASSER	see rpmsg	Assert implementation.

How to format rpmsg-lite code

To format code, use the application developed by Google, named *clang-format*. This tool is part of the llvm project. Currently, the clang-format 10.0.0 version is used for rpmsg-lite. The set of style settings used for clang-format is defined in the `.clang-format` file, placed in a root of the rpmsg-lite directory where Python script `run_clang_format.py` can be executed. This script executes the application named *clang-format.exe*. You need to have the path of this application in the OS's environment path, or you need to change the script.

References

[1] M. Novak, M. Cingel, Lockless Shared Memory Based Multicore Communication Protocol

Copyright © 2016 Freescale Semiconductor, Inc. Copyright © 2016-2025 NXP

Changelog RPMSG-Lite

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

v5.5.0

Added

- Added MCXE32B dual Cortex-M7 parts support.
- Added rpmsg_lite_are_all_buffers_consumed() API to report when all locally allocated TX buffers were consumed by the peer and returned to the local free buffer pool, with test coverage in tests/03_send_receive for both sides.
- Added i.MX937 parts support
- Removed obsolete unused k32w1 platform layer

v5.4.1

Fixed

- Corrected isr_counter3 handling in RT700 porting layer platform_deinit_interrupt implementation.

v5.4.0

Added

- Added i.MX93W parts support
- Added i.MX952 parts support
- Added mcxl14x parts support
- Added mcxw70 parts support
- Zephyr rpmsg-lite samples add support for RT600 and RT700 DSP cores

Changed

- Port platform_get_custom_shmem_config change return value from uint32_t to int32_t for error handling consistency.

Fixed

- Updated unit tests code to allow correct GCOV code coverage measurement.
- Added missing dsb instructions into all ISRs for cm4/cm7 based platform layers.
- Fixed CERT EXP34-C by adding __attribute__((noreturn)) to RL_HANG(void) function.
- **MISRA C-2012 Rule 14.3 compliance**: Simplified RL_WORD_ALIGN_UP and RL_WORD_ALIGN_DOWN macros to eliminate invariant controlling expression violations (CID 5011148)
  - Replaced ternary operator implementations with standard bitwise alignment idioms
  - Fixed hardcoded 4U constant in RL_WORD_ALIGN_UP to use RL_WORD_SIZE for better portability
- Fixed several MISRA compliance violations

v5.3.0

Added

- RT700 porting layer added support to send rpmsg messages between CM33_0 <-> Hifi1 and CM33_1 <-> Hifi4 cores.
- Add new platform macro RL_PLATFORM_MAX_ISR_COUNT this will set number of IRQ count per platform. This macro is then used in environment layers to set isr_table size where irq handles are registered. It size should match the bit length of VQ_ID so all combinations can fit into table.
- Unit tests updated to improve code coverage, new unit tests added covering static allocations in rtos environment layers.

Fixed

- virtio.h removed typedef uint8_t boolean and in its place use standard C99 bool type to avoid potential type conflicts.
- env_acquire_sync_lock() and env_release_sync_lock() synchronization primitives removed
- Kconfig consolidation, when RL_ALLOW_CUSTOM_SHMEM_CONFIG enabled the platform_get_custom_shmem_config() function needs to be implemented in platform layer to provide custom shared memory configuration for RPMsg-Lite instance.

v5.2.1

Added

- Doc added RPMSG-Lite VirtIO Overview
- Doc added RPSMG-Lite Design Considerations
- Added frdmimxrt1186 unit testing

Changed

- Remove limitation that RL_BUFFER_SIZE needs to be power of 2. It just has to be more than 16 bytes, e.g. 16 bytes of rpmsg header and payload size at least 1 byte and word aligned, if not it will be aligned up.

Fixed

- Fixed CERT-C INT31-C violation in platform_notify function in rpmsg_platform.c for imxrt700_m33, imxrt700_hifi4, imxrt700_hifi1 platforms

v5.2.0

Added

- Add MCXL20 porting layer and unit testing
- New utility macro RL_CALCULATE_BUFFER_COUNT_DOWN_SAFE to safely determine maximum buffer count within shared memory while preventing integer underflow.
- RT700 platform add support for MCMGR in DSPs

Changed

- Change rpmsg_platform.c to support new MCMGR API
- Improved input validation in initialization functions to properly handle insufficient memory size conditions.
- Refactored repeated buffer count calculation pattern for better code maintainability.

- To make sure that remote has already registered IRQ there is required App level IPC mechanism to notify master about it

Fixed

- Fixed env_wait_for_link_up function to handle timeout in link state checks for baremetal and qnx environment, RL_BLOCK mode can be used to wait indefinitely.
- Fixed CERT-C INT31-C violation by adding compile-time check to ensure RL_PLATFORM_HIGHEST_LINK_ID remains within safe range for 16-bit casting in virtqueue ID creation.
- Fixed CERT-C INT30-C violations by adding protection against unsigned integer underflow in shared memory calculations, specifically in shmem_length - (uint32_t)RL_VRING_OVERHEAD and shmem_length - 2U * shmem_config.vring_size expressions.
- Fixed CERT INT31-C violation in platform_interrupt_disable() and similar functions by replacing unsafe cast from uint32_t to int32_t with a return of 0 constant.
- Fixed unsigned integer underflow in rpmsg_lite_alloc_tx_buffer() where subtracting header size from buffer size could wrap around if buffer was too small, potentially leading to incorrect buffer sizing.
- Fixed CERT-C INT31-C violation in rpmsg_lite.c where size parameter was cast from uint32_t to uint16_t without proper validation.
  - Applied consistent masking approach to both size and flags parameters: (uint16_t)(value & 0xFFFFU).
  - This fix prevents potential data loss when size values exceed 65535.
- Fixed CERT INT31-C violation in env_memset functions by explicitly converting int32_t values to unsigned char using bit masking. This prevents potential data loss or misinterpretation when passing values outside the unsigned char range (0-255) to the standard memset() function.
- Fixed CERT-C INT31-C violations in RPMsg-Lite environment porting: Added validation checks for signed-to-unsigned integer conversions to prevent data loss and misinterpretation.
  - rpmsg_env_freertos.c: Added validation before converting int32_t to UBaseType_t.
  - rpmsg_env_qnx.c: Fixed format string and added validation before assigning to mqstat fields.
  - rpmsg_env_threadx.c: Added validation to prevent integer overflow and negative values.
  - rpmsg_env_xos.c: Added range checking before casting to uint16_t.
  - rpmsg_env_zephyr.c: Added validation before passing values to k_msgq_init.
- Fixed a CERT INT31-C compliance issue in env_get_current_queue_size() function where an unsigned queue count was cast to a signed int32_t without proper validation, which could lead to lost or misinterpreted data if queue size exceeded INT32_MAX.
- Fixed CERT INT31-C violation in rpmsg_platform.c where memcmp() return value (signed int) was compared with unsigned constant without proper type handling.
- Fixed CERT INT31-C violation in rpmsg_platform.c where casting from uint32_t to uint16_t could potentially result in data loss. Changed length variable type from uint16_t to uint32_t to properly handle memory address differences without truncation.
- Fixed potential integer overflow in env_sleep_msec() function in ThreadX environment implementation by rearranging calculation order in the sleep duration formula.

- Fixed CERT-C INT31-C violation in RPMsg-Lite where bitwise NOT operations on integer constants were performed in signed integer context before being cast to unsigned. This could potentially lead to misinterpreted data on imx943 platform.
- Added RL_MAX_BUFFER_COUNT (32768U) and RL_MAX_VRING_ALIGN (65536U) limit to ensure alignment values cannot contribute to integer overflow
- Fixed CERT INT31-C violation in vring_need_event(), added cast to uint16_t for each operand.

v5.1.4 - 27-Mar-2025

Added

- Add KW43B43 porting layer

Changed

- Doxygen bump to version 1.9.6

v5.1.3 - 13-Jan-2025

Added

- Memory cache management of shared memory. Enable with #define RL_USE_DCACHE (1) in rpmsg_config.h in case of data cache is used.
- Cmake/Kconfig support added.
- Porting layers for imx95, imxrt700, mcmxw71x, mcmxw72x, kw47b42 added.

v5.1.2 - 08-Jul-2024

Changed

- Zephyr-related changes.
- Minor Misra corrections.

v5.1.1 - 19-Jan-2024

Added

- Test suite provided.
- Zephyr support added.

Changed

- Minor changes in platform and env. layers, minor test code updates.

v5.1.0 - 02-Aug-2023

Added

- RPMsg-Lite: Added aarch64 support.

Changed

- RPMsg-Lite: Increased the queue size to (2 * RL_BUFFER_COUNT) to cover zero copy cases.

- Code formatting using LLVM16.

Fixed

- Resolved issues in ThreadX env. layer implementation.

v5.0.0 - 19-Jan-2023

Added

- Timeout parameter added to rpmsg_lite_wait_for_link_up API function.

Changed

- Improved debug check buffers implementation - instead of checking the pointer fits into shared memory check the presence in the VirtIO ring descriptors list.
- VRING_SIZE is set based on number of used buffers now (as calculated in vring_init) - updated for all platforms that are not communicating to Linux rpmsg counterpart.

Fixed

- Fixed wrong RL_VRING_OVERHEAD macro comment in platform.h files
- Misra corrections.

v4.0.0 - 20-Jun-2022

Added

- Added support for custom shared memory arrangement per the RPMsg_Lite instance.
- Introduced new rpmsg_lite_wait_for_link_up() API function - this allows to avoid using busy loops in rtos environments, GitHub PR #21.

Changed

- Adjusted rpmsg_lite_is_link_up() to return RL_TRUE/RL_FALSE.

v3.2.0 - 17-Jan-2022

Added

- Added support for i.MX8 MP multicore platform.

Changed

- Improved static allocations - allow OS-specific objects being allocated statically, GitHub PR #14.
- Aligned rpmsg_env_xos.c and some platform layers to latest static allocation support.

Fixed

- Minor Misra and typo corrections, GitHub PR #19, #20.

v3.1.2 - 16-Jul-2021

Added

- Addressed MISRA 21.6 rule violation in rpmsg_env.h (use SDK's PRINTF in MCUXpressoSDK examples, otherwise stdio printf is used).
- Added environment layers for XOS.
- Added support for i.MX RT500, i.MX RT1160 and i.MX RT1170 multicore platforms.

Fixed

- Fixed incorrect description of the rpmsg_lite_get_endpoint_from_addr function.

Changed

- Updated RL_BUFFER_COUNT documentation (issue #10).
- Updated imxrt600_hifi4 platform layer.

v3.1.1 - 15-Jan-2021

Added

- Introduced RL_ALLOW_CONSUMED_BUFFERS_NOTIFICATION config option to allow opposite side notification sending each time received buffers are consumed and put into the queue of available buffers.
- Added environment layers for Threadx.
- Added support for i.MX8QM multicore platform.

Changed

- Several MISRA C-2012 violations addressed.

v3.1.0 - 22-Jul-2020

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed (7.4).
- Fixed missing lock in rpmsg_lite_rx_callback() for QNX env.
- Correction of rpmsg_lite_instance structure members description.
- Address -Waddress-of-packed-member warnings in GCC9.

Changed

- Clang update to v10.0.0, code re-formatted.

v3.0.0 - 20-Dec-2019

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed, incl. data types consolidation.

- Code formatted.

v2.2.0 - 20-Mar-2019

Added

- Added configuration macro RL_DEBUG_CHECK_BUFFERS.
- Several MISRA violations fixed.
- Added environment layers for QNX and Zephyr.
- Allow environment context required for some environment (controlled by the RL_USE_ENVIRONMENT_CONTEXT configuration macro).
- Data types consolidation.

v1.1.0 - 28-Apr-2017

Added

- Supporting i.MX6SX and i.MX7D MPU platforms.
- Supporting LPC5411x MCU platform.
- Baremental and FreeRTOS support.
- Support of copy and zero-copy transfer.
- Support of static API (without dynamic allocations).

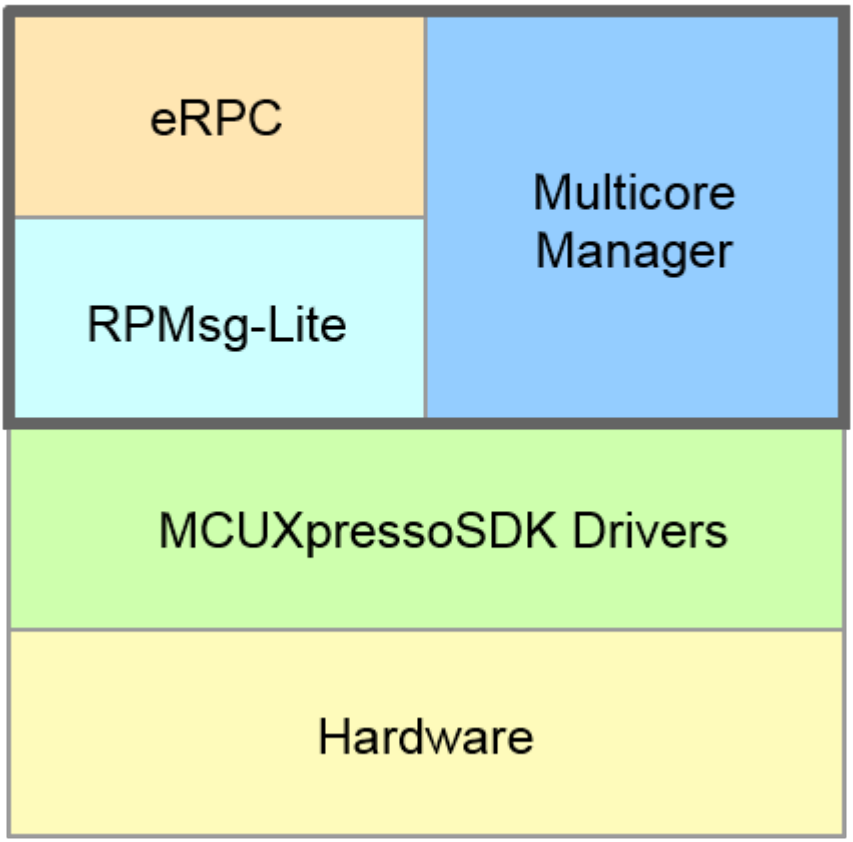
Multicore Manager (MCMGR)

Quick links

- *Changelog*
- API documentation (Doxygen): https://nxp-mcuxpresso.github.io/mcux-mcmgr/

The Multicore Manager (MCMGR) software library provides a number of services for multicore systems. This library is distributed as a part of the Multicore SDK (MCSDK). Together, the MCSDK and the MCUXpresso SDK (SDK) form a framework for development of software for NXP multicore devices.

The MCMGR component is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr directory.



The Multicore Manager provides the following major functions:

- Maintains information about all cores in system.
- Secondary/auxiliary core(s) startup and shutdown.
- Remote core monitoring and event handling.

Usage of the MCMGR software component

The main use case of MCMGR is the secondary/auxiliary core start. This functionality is performed by the public API function.

Example of MCMGR usage to start secondary core:

```
#include "mcmgr.h"

void main()
{
    /* Initialize MCMGR, low level multicore management library */
    MCMGR_Init();

    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass "1" as startup data,␣
↪starting synchronously. */
    MCMGR_StartCore(kMCMGR_Core1, CORE1_BOOT_ADDRESS, 1, kMCMGR_Start_Synchronous);
```

(continues on next page)

(continued from previous page)

```
.
.
.
    /* Stop secondary core execution. */
    MCMGR_StopCore(kMCMGR_Core1);
}
```

Some platforms allow stopping and re-starting the secondary core application again, using the MCMGR_StopCore / MCMGR_StartCore API calls. It is necessary to ensure the initially loaded image is not corrupted before re-starting, especially if it deals with the RAM target. Cache coherence has to be considered/ensured as well.

It could also happen that the secondary core application stops running correctly and the primary core application does not know about that situation. Therefore, it is beneficial to implement a mechanism for core health monitoring. The test_heartbeat unit test can serve as an example how to ensure that: secondary core could periodically send heartbeat signals to the primary core using MCMGR_TriggerEvent() API to indicate that it is alive and functioning properly.

Another important MCMGR feature is the ability for remote core monitoring and handling of events such as reset, exception, and application events. Application-specific callback functions for events are registered by the MCMGR_RegisterEvent() API. Triggering these events is done using the MCMGR_TriggerEvent() API. mcmgr_event_type_t enums all possible event types.

An example of MCMGR usage for remote core monitoring and event handling. Code for the primary side:

```
#include "mcmgr.h"

#define APP_RPMSG_READY_EVENT_DATA  (1)
#define APP_NUMBER_OF_CORES (2)
#define APP_SECONDARY_CORE kMCMGR_Core1

/* Callback function registered via the MCMGR_RegisterEvent() and triggered by MCMGR_TriggerEvent()␣
↪called on the secondary core side */
void RPMsgRemoteReadyEventHandler(mcmgr_core_t coreNum, uint16_t eventData, void *context)
{
    uint16_t *data = &((uint16_t *)context)[coreNum];

    *data = eventData;
}

void main()
{
    uint16_t RPMsgRemoteReadyEventData[NUMBER_OF_CORES] = {0};

    /* Initialize MCMGR, low level multicore management library */
    MCMGR_Init();

    /* Register the application event before starting the secondary core */
    MCMGR_RegisterEvent(kMCMGR_RemoteApplicationEvent, RPMsgRemoteReadyEventHandler, (void␣
↪*)RPMsgRemoteReadyEventData);

    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass rpmsg_lite_base address␣
↪as startup data, starting synchronously. */
    MCMGR_StartCore(APP_SECONDARY_CORE, CORE1_BOOT_ADDRESS, (uint32_t)rpmsg_lite_
↪base, kMCMGR_Start_Synchronous);

    /* Wait until the secondary core application signals the rpmsg remote has been initialized and is ready to␣
↪communicate. */
    while(APP_RPMSG_READY_EVENT_DATA != RPMsgRemoteReadyEventData[APP_SECONDARY_
↪CORE]) {};
```

(continues on next page)

(continued from previous page)

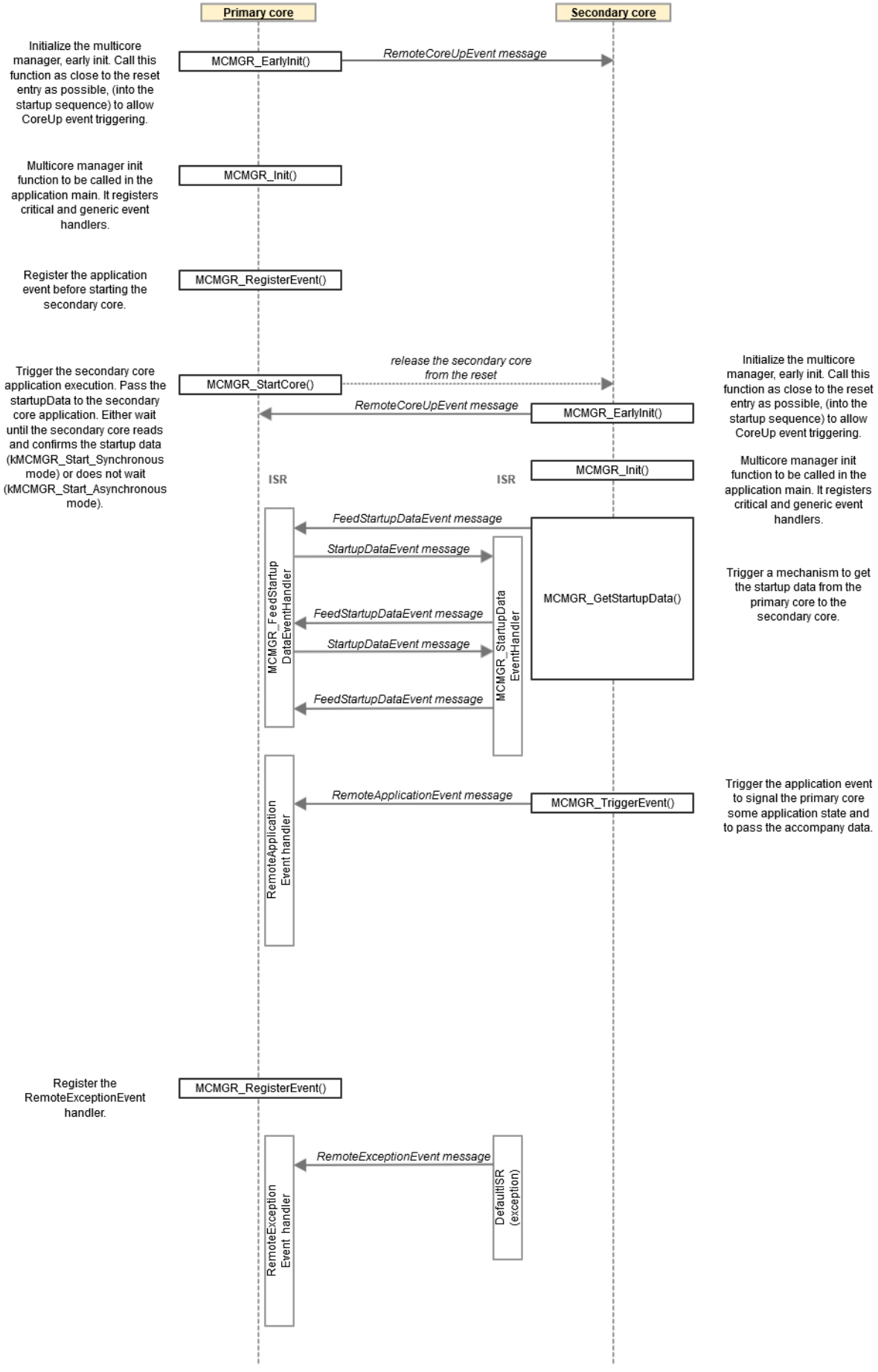
```
.
.
.
}
```

Code for the secondary side:

```
#include "mcmgr.h"

#define APP_RPMSG_READY_EVENT_DATA  (1)

void main()
{
    /* Initialize MCMGR, low level multicore management library */
    MCMGR_Init();
.
.
.

    /* Signal the to other core that we are ready by triggering the event and passing the APP_RPMSG_
↪READY_EVENT_DATA */
    MCMGR_TriggerEvent(kMCMGR_Core0, kMCMGR_RemoteApplicationEvent, APP_RPMSG_
↪READY_EVENT_DATA);
.
.
.
}
```

MCMGR Data Exchange Diagram

The following picture shows how the handshakes are supposed to work between the two cores in the MCMGR software.


Primary core
Secondary core
Initialize the multicore manager, early init. Call this function as close to the reset entry as possible, (into the startup sequence) to allow CoreUp event triggering.
MCMGR_EarlyInit()
RemoteCoreUpEvent message
Multicore manager init function to be called in the application main. It registers critical and generic event handlers.
MCMGR_Init()
Register the application event before starting the secondary core.
MCMGR_RegisterEvent()
Trigger the secondary core application execution. Pass the startupData to the secondary core application. Either wait until the secondary core reads and confirms the startup data (kMCMGR_Start_Synchronous mode) or does not wait (kMCMGR_Start_Asynchronous mode).
MCMGR_StartCore()
release the secondary core from the reset
RemoteCoreUpEvent message
MCMGR_EarlyInit()
Initialize the multicore manager, early init. Call this function as close to the reset entry as possible, (into the startup sequence) to allow CoreUp event triggering.
MCMGR_Init()
Multicore manager init function to be called in the application main. It registers critical and generic event handlers.
ISR
ISR
FeedStartupDataEvent message
StartupDataEvent message
FeedStartupDataEvent message
StartupDataEvent message
FeedStartupDataEvent message
MCMGR_FeedStartup DataEventHandler
MCMGR_StartupData EventHandler
MCMGR_GetStartupData()
Trigger a mechanism to get the startup data from the primary core to the secondary core.
RemoteApplicationEvent message
MCMGR_TriggerEvent()
RemoteApplication Event handler
Trigger the application event to signal the primary core some application state and to pass the accompany data.
Register the RemoteExceptionEvent handler.
MCMGR_RegisterEvent()
RemoteExceptionEvent message
RemoteException Event handler
DefaultISR (exception)

Changelog Multicore Manager

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

v5.3.0

Added

- Added MCXE32B (dual Cortex-M7) porting layer.
- Remove obsolete unused platform layers
- Added initialization guard to public MCMGR API functions (RPSDK-1183): guarded functions return kStatus_MCMGR_NotReady before MCMGR_Init() completes.

Fixed

- MCUX-88924: Fix KW43 ICS init-handshake hang (RPSDK-1183 regression): s_mcmgrInitialized is now set before mcmgr_platform_init_internal() enables the MU RX IRQ, so re-entrant ISR dispatch during MCMGR_Init() is no longer silently dropped by MCMGR_CHECK_INIT().
- MCUX-89876: Fix Coverity DEADCODE and UNUSED_VALUE findings in mcmgr_internal_core_api_imxrt700.c: align mcmgr_mu_channel_handler event-range guard with positive-logic form (matching imxrt1170), and aggregate mcmgr_trigger_event_internal return values in mcmgr_platform_init_internal_early so earlier errors are not silently overwritten.

v5.2.0

Changed

- Merged internal platform initialization into a single $\mathrm{mcmgr_platform_init_internal()}$ function.
- Simplified initialization API: $\mathrm{MCMGR_Init()}$ now handles both early and standard initialization internally.

Deprecated

- $\mathrm{MCMGR_EarlyInit()}$ is now deprecated and a no-op. It remains for backward compatibility and may be removed in a future major version. Replace calls to $\mathrm{MCMGR_EarlyInit()}$ + $\mathrm{MCMGR_Init()}$ with a single $\mathrm{MCMGR_Init()}$ call.

Migration Guide

Before:

```
MCMGR_EarlyInit();
// ...
MCMGR_Init();
```

After:

```
MCMGR_Init();
```

v5.1.0

Added

- Added mcxl14x parts support
- Added mcxw70 parts support
- Add new config MCMGR_REMOTE_APP_EVENT_COUNT for configurable remote application event count. This will expand number of application events available for MCMGR users. kMCMGR_RemoteApplicationEvent, kMCMGR_RemoteApplicationEvent1 .. kMCMGR_RemoteApplicationEvent4.
- Align Kinetis parts porting to changed register map

Fixed

- Fix MISRA C-2012 Rule 20.8 violations.

v5.0.2

Added

- Added gcov options and configs to support mcmgr code coverage
- Added new test_weak_mu_isr testcase for devices with MU peripheral
- Added new test_heartbeat testcase showing heartbeat mechanism between primary and secondary cores using the MCMGR

v5.0.1

Added

- Added frdmimxrt1186 unit testing

Changed

- [KW43] Rename core#1 reset control register

Fixed

- Added CX flag into CMakeLists.txt to allow c++ build compatibility.
- Fix path to mcmgr headers directory in doxyfile

v5.0.0

Added

- Added MCMGR_BUSY_POLL_COUNT macro to prevent infinite polling loops in MCMGR operations.
- Implemented timeout mechanism for all polling loops in MCMGR code.
- Added support to handle more then two cores. Breaking API change by adding parameter coreNum specifying core number in functions bellow.
  - MCMGR_GetStartupData(uint32_t *startupData, mcmgr_core_t coreNum)
  - MCMGR_TriggerEvent(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)
  - MCMGR_TriggerEventForce(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)
  - typedef void (*mcmgr_event_callback_t)(uint16_t data, void *context, mcmgr_core_t coreNum);

When registering the event with function MCMGR_RegisterEvent() user now needs to provide callbackData pointer to array of elements per every core in system (see README.md for example).In case of systems with only two cores the coreNum in callback can be ignored as events can arrive only from one core. Please see Porting guide for more details: Porting-GuideTo_v5.md

- Updated all porting files to support new MCMGR API.
- Added new platform specific include file mcmgr_platform.h. It will contain common platform specific macros that can be then used in mcmgr and application. e.g. platform core count MCMGR_CORECOUNT 4.
- Move all header files to new inc directory.
- Added new platform-specific include files inc/platform/<platform_name>/mcmgr_platform.h.

Added

- Add MCXL20 porting layer and unit testing

v4.1.7

Fixed

- mcmgr_stop_core_internal() function now returns kStatus_MCMGR_NotImplemented status code instead of kStatus_MCMGR_Success when device does not support stop of secondary core. Ports affected: kw32w1, kw45b41, kw45b42, mcxw716, mcxw727.

[v4.1.6]

Added

- Multicore Manager moved to standalone repository.
- Add porting layers for imxrt700, mcmxw727, kw47b42.
- New MCMGR_ProcessDeferredRxIsr() API added.

[v4.1.5]

Added

- Add notification into MCMGR_EarlyInit and mcmgr_early_init_internal functions to avoid using uninitialized data in their implementations.

[v4.1.4]

Fixed

- Avoid calling tx isr callbacks when respective Messaging Unit Transmit Interrupt Enable flag is not set in the CR/TCR register.
- Messaging Unit RX and status registers are cleared after the initialization.

[v4.1.3]

Added

- Add porting layers for imxrt1180.

Fixed

- mu_isr() updated to avoid calling tx isr callbacks when respective Transmit Interrupt Enable flag is not set in the CR/TCR register.
- mcmgr_mu_internal.c code adaptation to new supported SoCs.

[v4.1.2]

Fixed

- Update mcmgr_stop_core_internal() implementations to set core state to kMCMGR_ResetCoreState.

[v4.1.0]

Fixed

- Code adjustments to address MISRA C-2012 Rules

[v4.0.3]

Fixed

- Documentation updated to describe handshaking in a graphic form.
- Minor code adjustments based on static analysis tool findings

[v4.0.2]

Fixed

- Align porting layers to the updated MCUXpressoSDK feature files.

[v4.0.1]

Fixed

- Code formatting, removed unused code

[v4.0.0]

Added

- Add new MCMGR_TriggerEventForce() API.

[v3.0.0]

Removed

- Removed MCMGR_LoadApp(), MCMGR_MapAddress() and MCMGR_SignalReady()

Modified

- Modified MCMGR_GetStartupData()

Added

- Added MCMGR_EarlyInit(), MCMGR_RegisterEvent() and MCMGR_TriggerEvent()
- Added the ability for remote core monitoring and event handling

[v2.0.1]

Fixed

- Updated to be Misra compliant.

[v2.0.0]

Added

- Support for lpcxpresso54114 board.

[v1.1.0]

Fixed

- Ported to KSDK 2.0.0.

[v1.0.0]

Added

- Initial release.

eRPC

eRPC (Embedded RPC) is an open-source remote procedure call (RPC) system suitable for multi-core and multiprocessor embedded systems.

Quick links

- *Getting Started*
- *Changelog*
- Upstream project: https://github.com/EmbeddedRPC/erpc

eRPC Getting Started

Overview

This *Getting Started User Guide* shows software developers how to use Remote Procedure Calls (RPC) in embedded multicore microcontrollers (eRPC).

The eRPC documentation is located in the *<MCUXpressoSDK_install_dir>/ middleware/multicore/erpc/doc* folder.

Create an eRPC application

This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.

2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
    1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
    2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
    3. Add infrastructure files.
    4. Add user code for client and server applications.
    5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

Multicore server application

The “Matrix multiply” eRPC server project is located in the following folder:

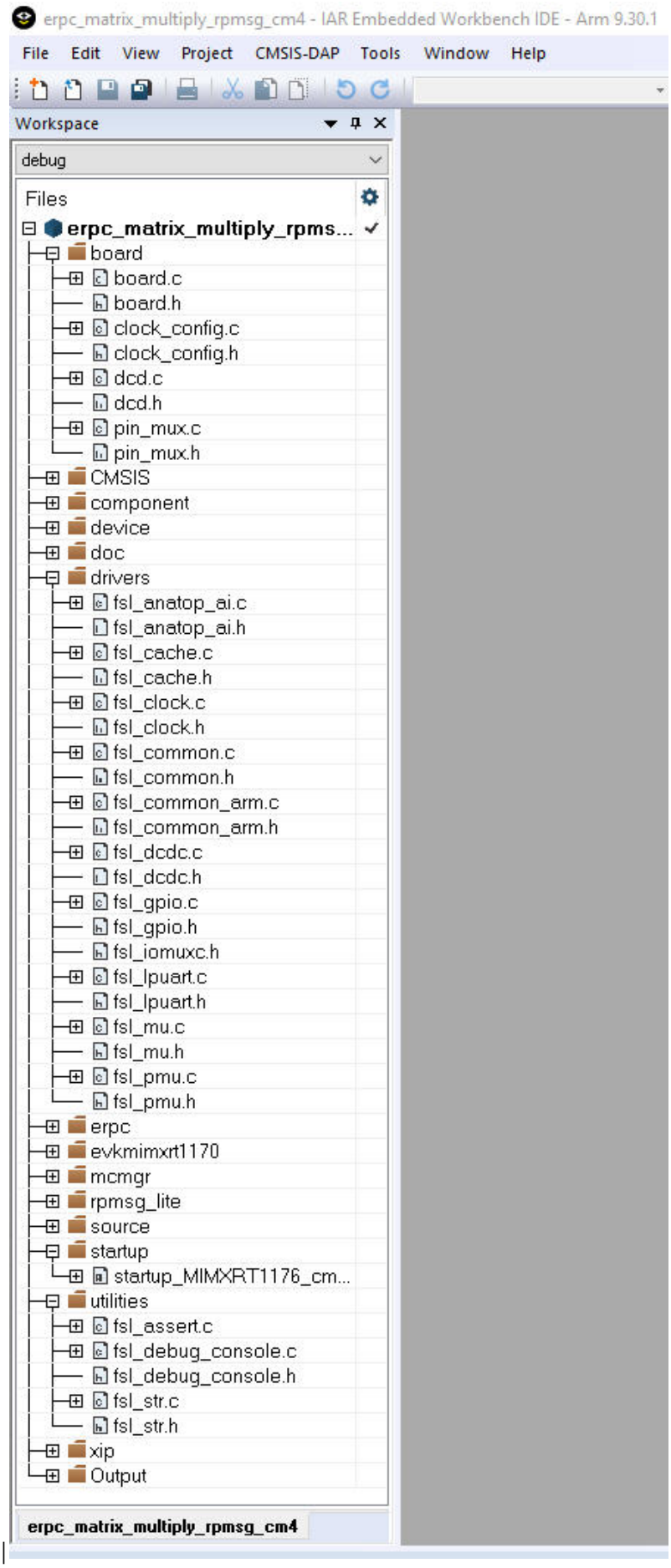
<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar

The project files for the eRPC server have the $_\mathrm{cm4}$ suffix.

Server project basic source files

The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- *<MCUXpressoSDK_install_dir>/devices/<device>*
- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

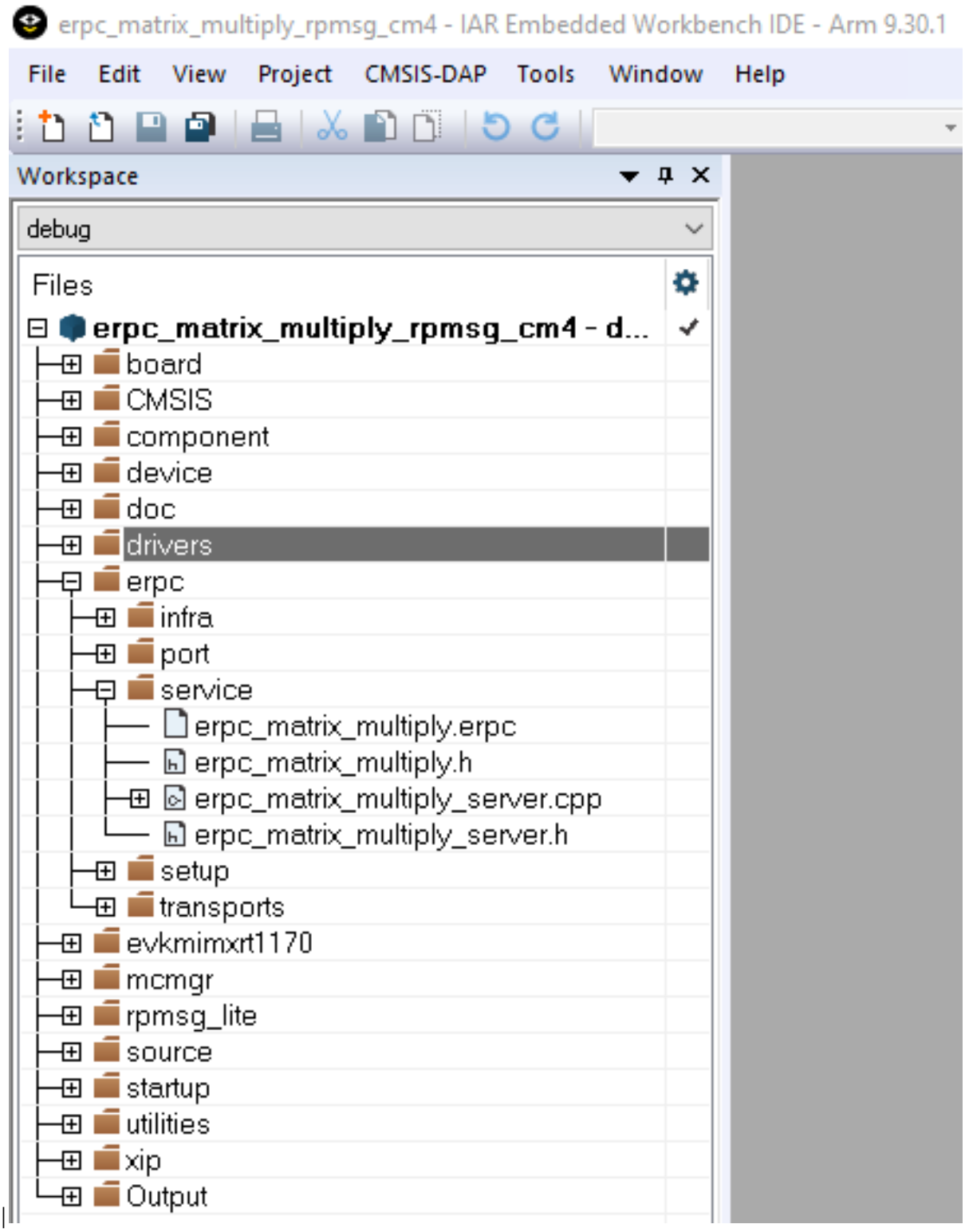
Server related generated files

The server-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/s



Parent topic: Multicore server application

Server infrastructure files

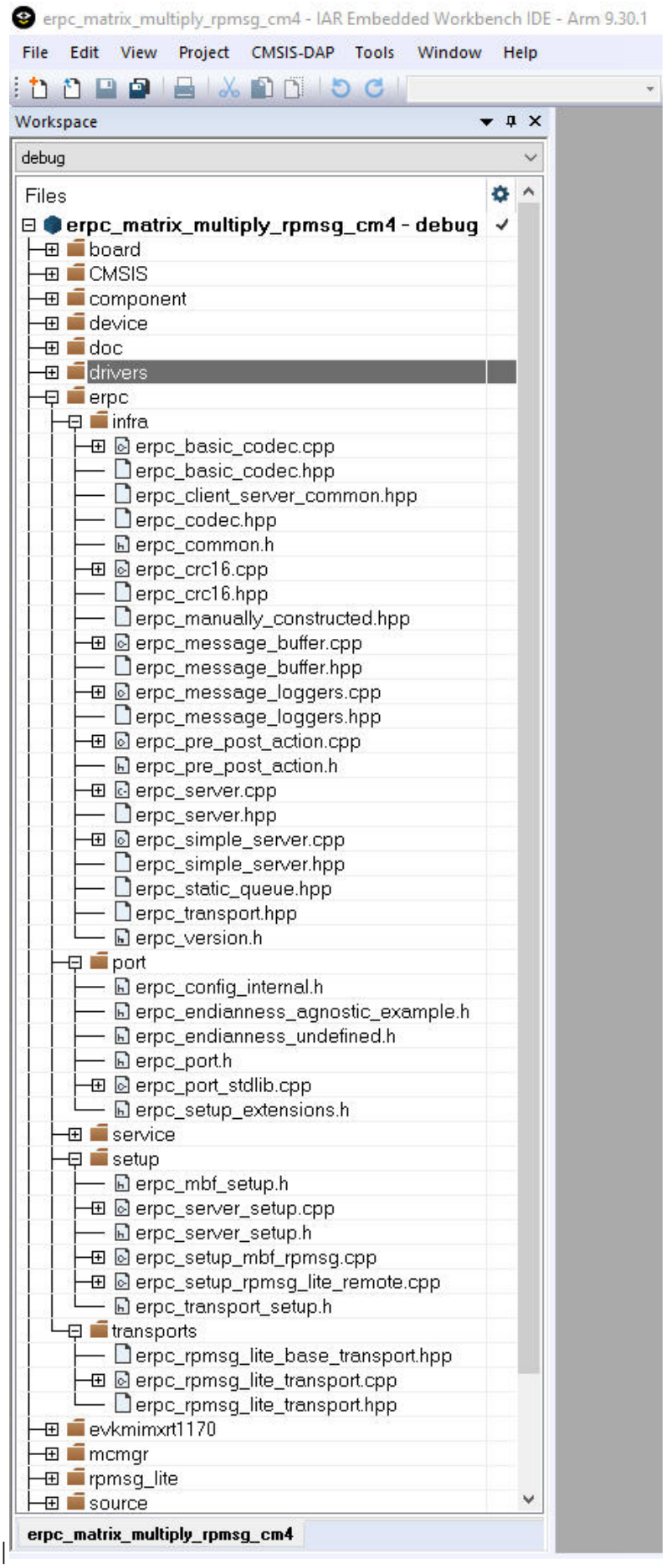
The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the *C/C++*

language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, erpc_server.hpp, erpc_server.cpp, erpc_simple_server.hpp, and erpc_simple_server.cpp, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The erpc_common.hpp file is used for common eRPC definitions, typedefs, and enums.
  - The erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: erpc_message_buffer.h and erpc_message_buffer.cpp.
  - The erpc_transport.h file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
  - erpc_port_stdlib.cpp file ensures adaptation to stdlib.
  - erpc_config_internal.h internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The erpc_server_setup.h and erpc_server_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
  - The erpc_transport_setup.h and erpc_setup_rpmsg_lite_remote.cpp files needs to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files need to be added into the server project.


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_server.cpp
erpc_server.hpp
erpc_simple_server.cpp
erpc_simple_server.hpp
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_mbf_setup.h
erpc_server_setup.cpp
erpc_server_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_remote.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

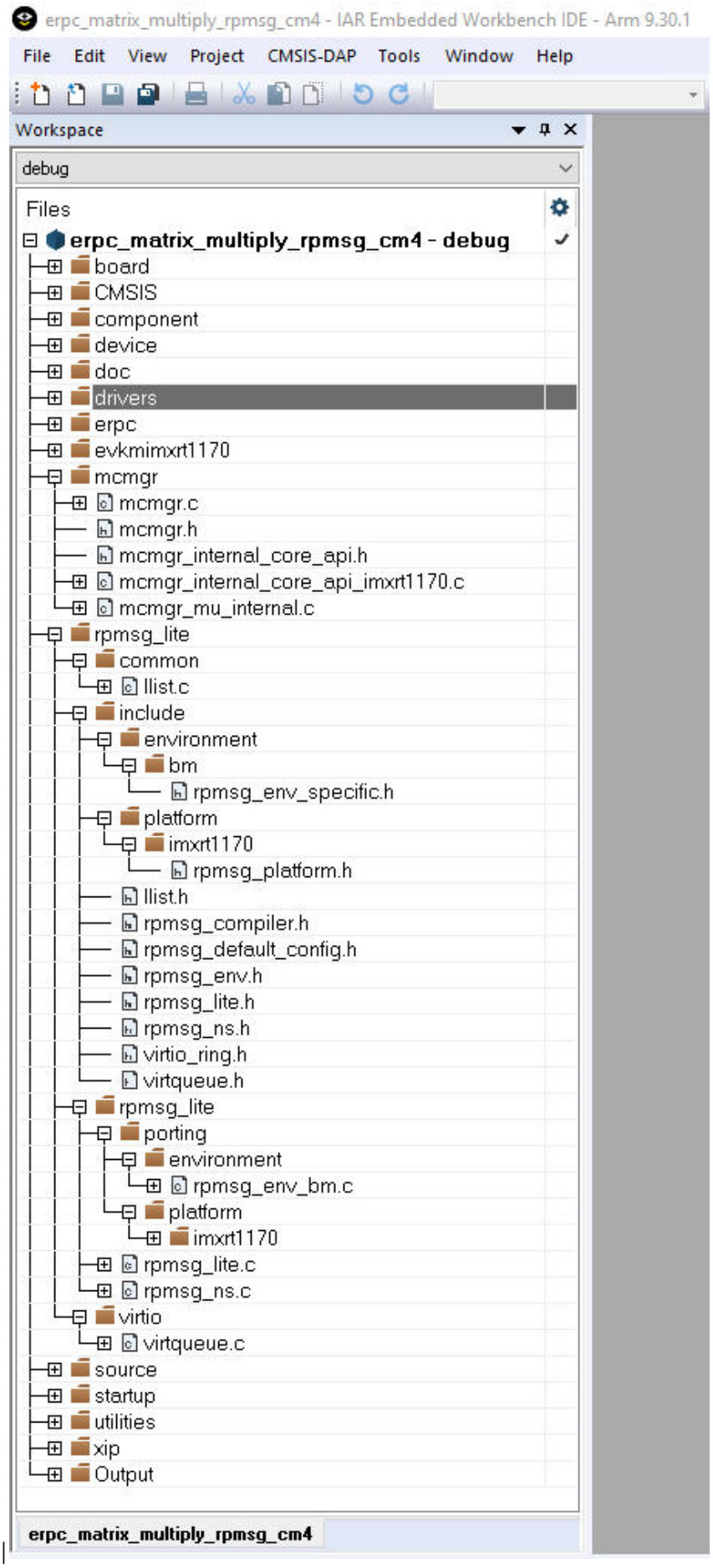
Server multicore infrastructure files

Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

Server user code

The server's user code is stored in the main_core1.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4

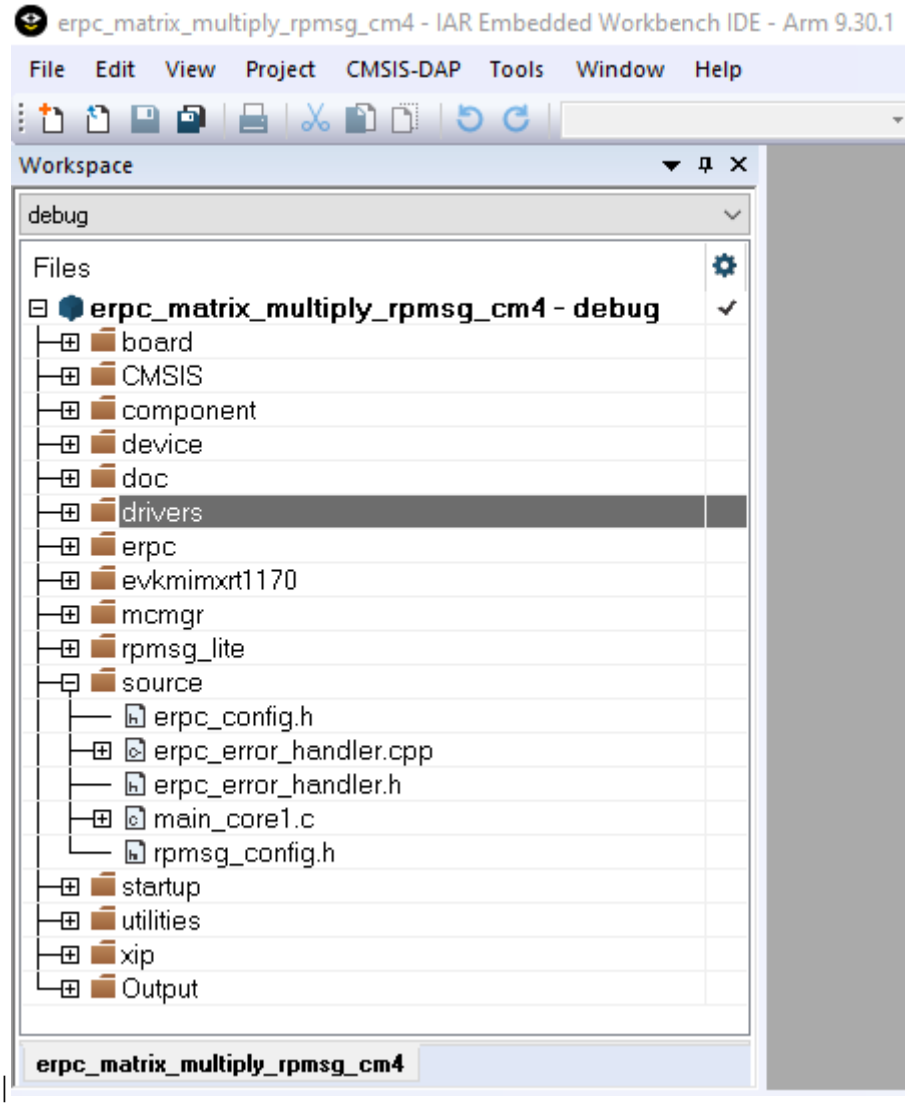
The main_core1.c file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the erpc_error_handler.h and erpc_error_handler.cpp files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
  ...
}
int main()
{
  ...
  /* RPMsg-Lite transport layer initialization */
  erpc_transport_t transport;
  transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
  ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_rpmsg_init(transport);
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server);
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg* folder.



Parent topic: Multicore server application

Parent topic:*Create an eRPC application*

Multicore client application

The “Matrix multiply” eRPC client project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar

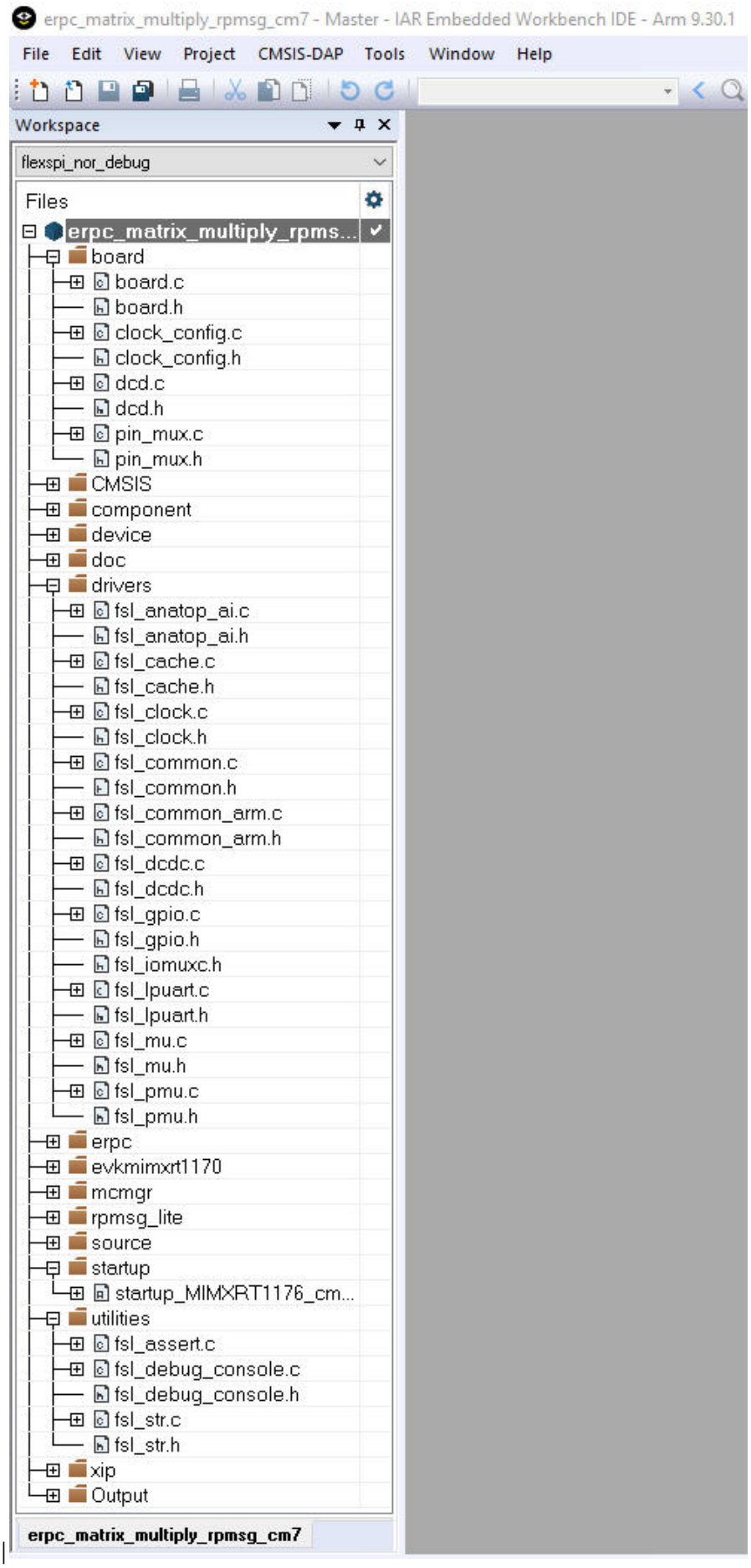
Project files for the eRPC client have the _cm7 suffix.

Client project basic source files

The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- *<MCUXpressoSDK_install_dir>/devices/<device>*

- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

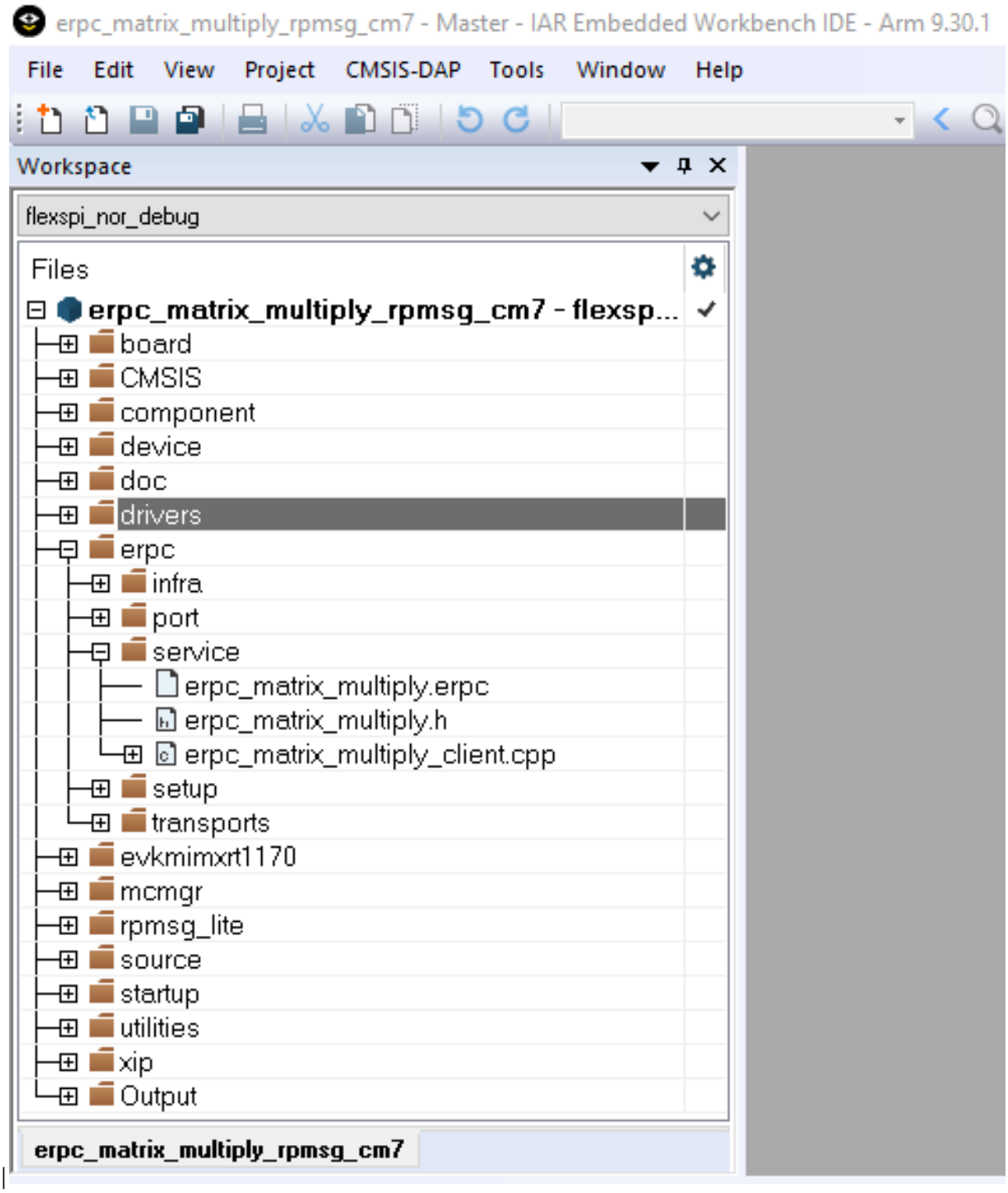
Parent topic: Multicore client application

Client-related generated files

The client-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/* folder.



Parent topic: Multicore client application

Client infrastructure files

The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, erpc_client_manager.h and erpc_client_manager.cpp, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- erpc_common.h file is used for common eRPC definitions, typedefs, and enums.
- erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: erpc_message_buffer.hpp and erpc_message_buffer.cpp.
- erpc_transport.hpp file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

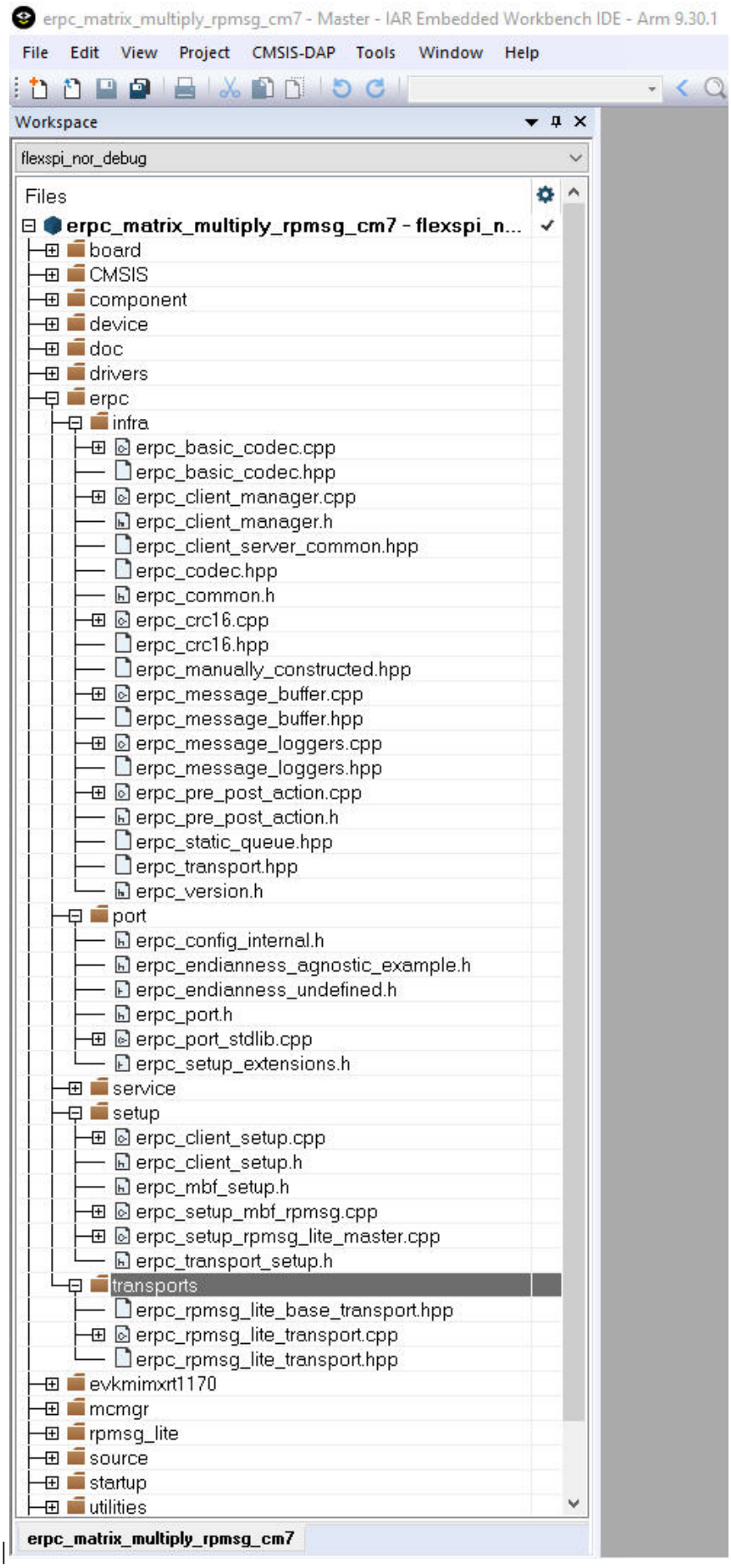
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- erpc_client_setup.h and erpc_client_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
- erpc_transport_setup.h and erpc_setup_rpmsg_lite_master.cpp files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files needs to be added into the client project.


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_n...
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_manager.cpp
erpc_client_manager.h
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_client_setup.cpp
erpc_client_setup.h
erpc_mbf_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_master.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
utilities
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

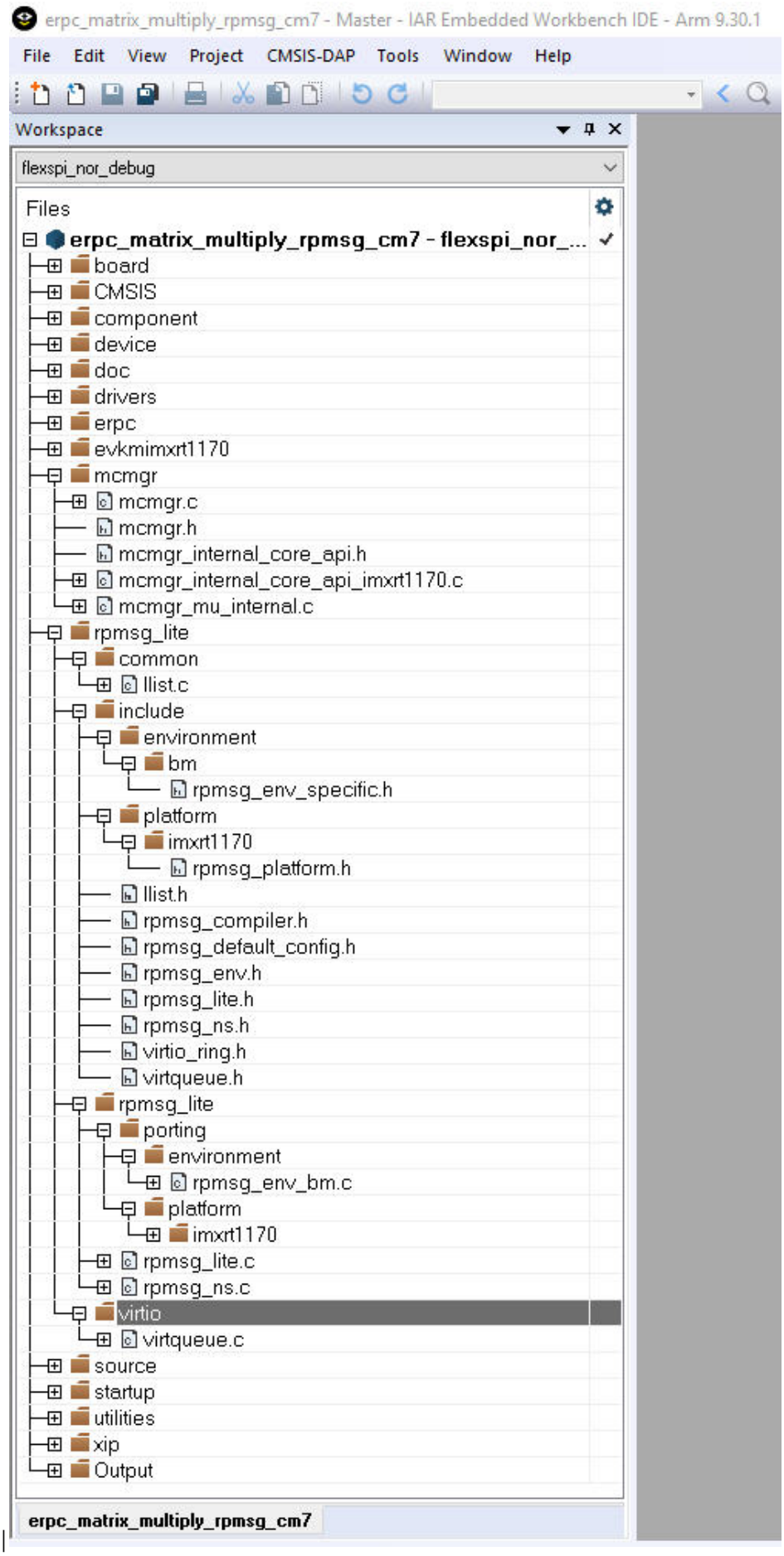
Client multicore infrastructure files

Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_nor_...
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

Client user code

The client's user code is stored in the main_core0.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_example/erpc_matrix_multiply_rpmsg/cm7

The main_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc_error_handler.h and erpc_error_handler.cpp files.

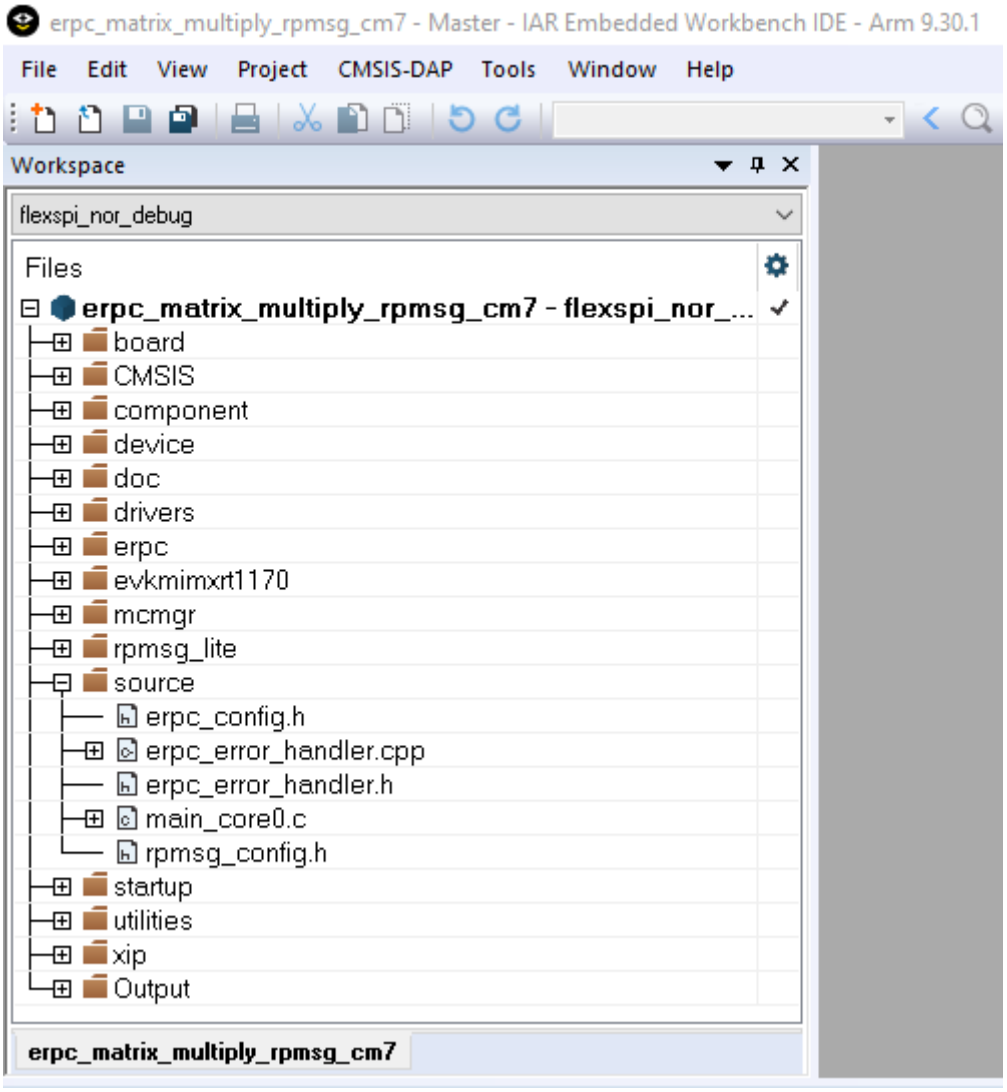
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg



|

|

Parent topic: Multicore client application

Parent topic:*Create an eRPC application*

Multiprocessor server application

The “Matrix multiply” eRPC server project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir»/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Server user code

The server's user code is stored in the main_server.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_lay* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
  ...
}
int main()
{
  ...
  /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
  erpc_transport_t transport;
  transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_dynamic_init();
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server)
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Parent topic: Multiprocessor server application

Multiprocessor client application

The "Matrix multiply" eRPC client project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<tr* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.hpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Client user code

The client's user code is stored in the main_client.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

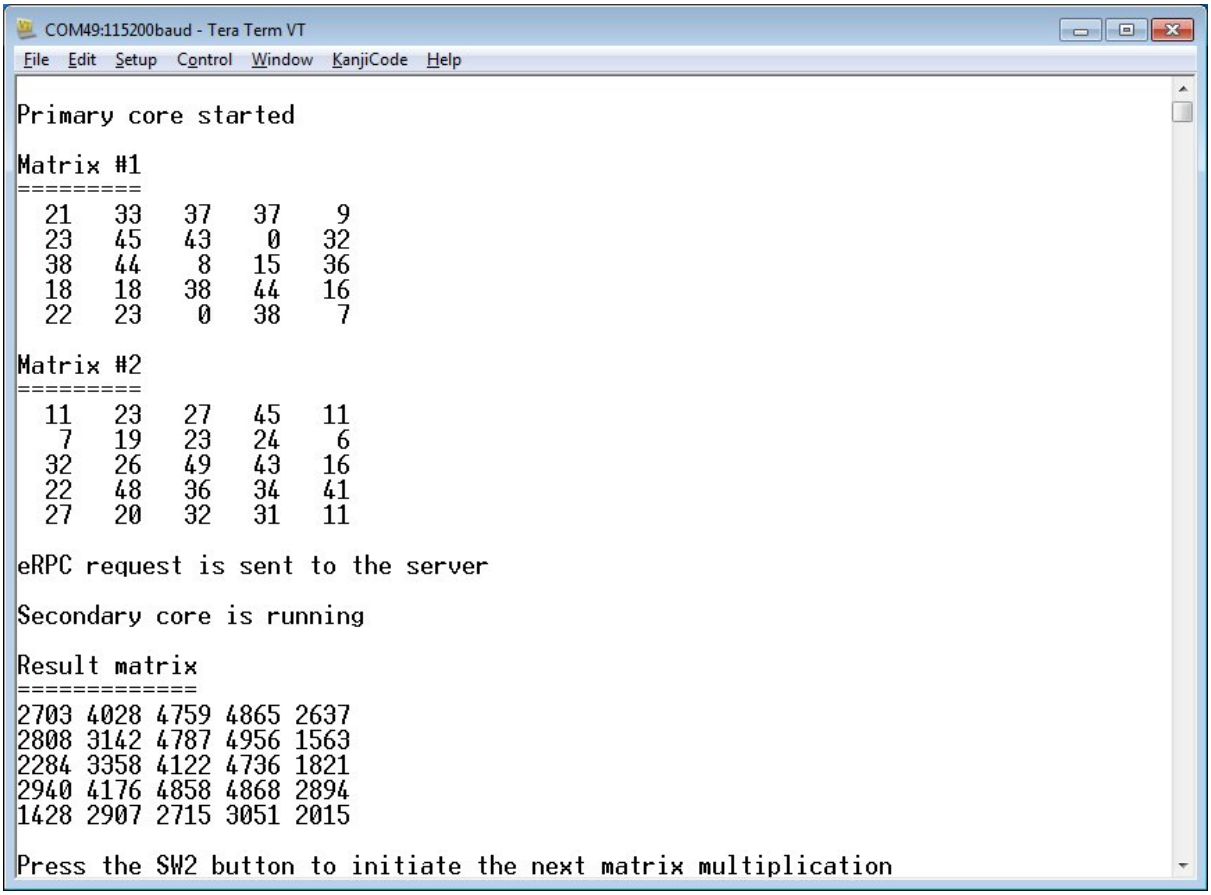
Parent topic: Multiprocessor client application

Parent topic: Multiprocessor server application

Parent topic:*Create an eRPC application*

Running the eRPC application

Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the <*MCUXpressoSDK_install_dir*>/*docs* folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:*Create an eRPC application*

Parent topic:*eRPC example*

eRPC example

This section shows how to create an example eRPC application called “Matrix multiply”, which implements one eRPC function (matrix multiply) with two function parameters (two matrices). The client-side application calls this eRPC function, and the server side performs the multiplication of received matrices. The server side then returns the result.

For example, use the NXP MIMXRT1170-EVK board as the target dual-core platform, and the IAR Embedded Workbench for ARM (EWARM) as the target IDE for developing the eRPC example.

- The primary core (CM7) runs the eRPC client.
- The secondary core (CM4) runs the eRPC server.
- RPMsg-Lite (Remote Processor Messaging Lite) is used as the eRPC transport layer.

The “Matrix multiply” application can be also run in the multi-processor setup. In other words, the eRPC client running on one SoC comunicates with the eRPC server that runs on anothe SoC, utilizing different transport channels. It is possible to run the board-to-PC example (PC as the eRPC server and a board as the eRPC client, and vice versa) and also the board-to-board example. These multiprocessor examples are prepared for selected boards only.

|Multicore application source and project files|*<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore*
|Multiprocessor application source and project files|*<MCUXpressoSDK_install_dir>/boards/<board_name>/multi*

<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<tr

| |eRPC source files|*<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/*| |RPMsg-Lite source files|*<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/*|

Designing the eRPC application

The matrix multiply application is based on calling single eRPC function that takes 2 two-dimensional arrays as input and returns matrix multiplication results as another 2 two-dimensional array. The IDL file syntax supports arrays with the dimension length set by the number only (in the current eRPC implementation). Because of this, a variable is declared in the IDL dedicated to store information about matrix dimension length, and to allow easy maintenance of the user and server code.

For a simple use of the two-dimensional array, the alias name (new type definition) for this data type has is declared in the IDL. Declaring this alias name ensures that the same data type can be used across the client and server applications.

Parent topic:*eRPC example*

Creating the IDL file

The created IDL file is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/s

The created IDL file contains the following code:

```
program erpc_matrix_multiply
/*! This const defines the matrix size. The value has to be the same as the
Matrix array dimension. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
const int32 matrix_size = 5;
/*! This is the matrix array type. The dimension has to be the same as the
matrix size const. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
type Matrix = int32[matrix_size][matrix_size];
interface MatrixMultiplyService {
erpcMatrixMultiply(in Matrix matrix1, in Matrix matrix2, out Matrix result_matrix) ->
void
}
```

Details:

- The IDL file starts with the program name (*erpc_matrix_multiply*), and this program name is used in the naming of all generated outputs.
- The declaration and definition of the constant variable named *matrix_size* follows next. The *matrix_size* variable is used for passing information about the length of matrix dimensions to the client/server user code.
- The alias name for the two-dimensional array type (*Matrix*) is declared.
- The interface group *MatrixMultiplyService* is located at the end of the IDL file. This interface group contains only one function declaration *erpcMatrixMultiply*.

- As shown above, the function's declaration contains three parameters of Matrix type: *matrix1* and *matrix2* are input parameters, while *result_matrix* is the output parameter. Additionally, the returned data type is declared as void.

When writing the IDL file, the following order of items is recommended:

1. Program name at the top of the IDL file.
2. New data types and constants declarations.
3. Declarations of interfaces and functions at the end of the IDL file.

Parent topic:*eRPC example*

Using the eRPC generator tool

|Windows OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Windows|
|Linux OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x64

<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x86

| |Mac OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Mac|

The files for the "Matrix multiply" example are pre-generated and already a part of the application projects. The following section describes how they have been created.

- The easiest way to create the shim code is to copy the erpcgen application to the same folder where the IDL file (*.erpc) is located; then run the following command:

  erpcgen <IDL_file>.erpc

- In the "Matrix multiply" example, the command should look like:

  erpcgen erpc_matrix_multiply.erpc

Additionally, another method to create the shim code is to execute the eRPC application using input commands:

- "-?"/"—help" – Shows supported commands.
- "-o <filePath>"/"—output<filePath>" – Sets the output directory.

For example,

```
<path_to_erpcgen>/erpcgen –o <path_to_output>
<path_to_IDL>/<IDL_file_name>.erpc
```

For the "Matrix multiply" example, when the command is executed from the default erpcgen location, it looks like:

erpcgen –o

../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service
../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/erpc_matrix_mu

In both cases, the following four files are generated into the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service folder.*

- erpc_matrix_multiply.h
- erpc_matrix_multiply_client.cpp
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

For multiprocessor examples, the eRPC file and pre-generated files can be found in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_common/erpc_matrix_multiply/serv* folder.

For Linux OS users:

- Do not forget to set the permissions for the eRPC generator application.
- Run the application as ./erpcgen... instead of as erpcgen ....

Parent topic:*eRPC example*

Create an eRPC application

This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
    1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
    2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
    3. Add infrastructure files.
    4. Add user code for client and server applications.
    5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

Multicore server application

The “Matrix multiply” eRPC server project is located in the following folder:

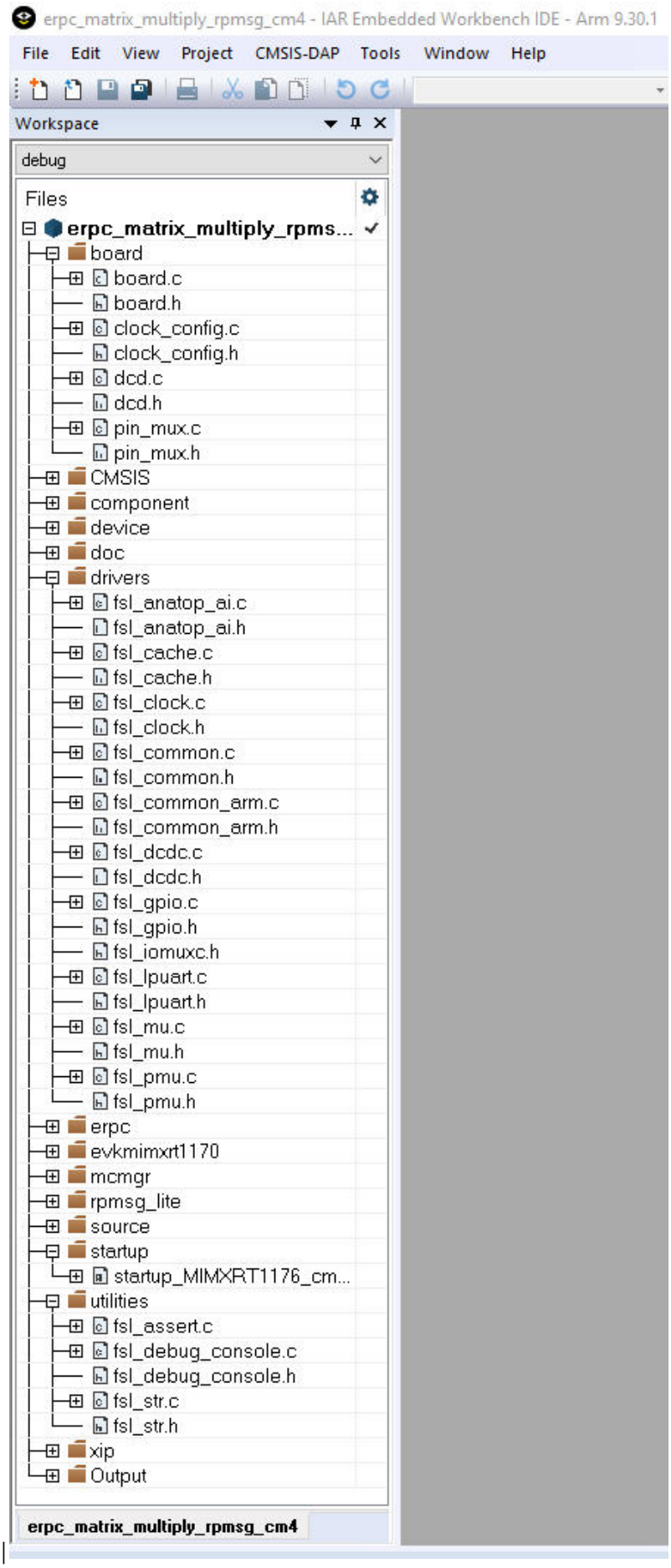
<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar

The project files for the eRPC server have the _cm4 suffix.

Server project basic source files

The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- *<MCUXpressoSDK_install_dir>/devices/<device>*
- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

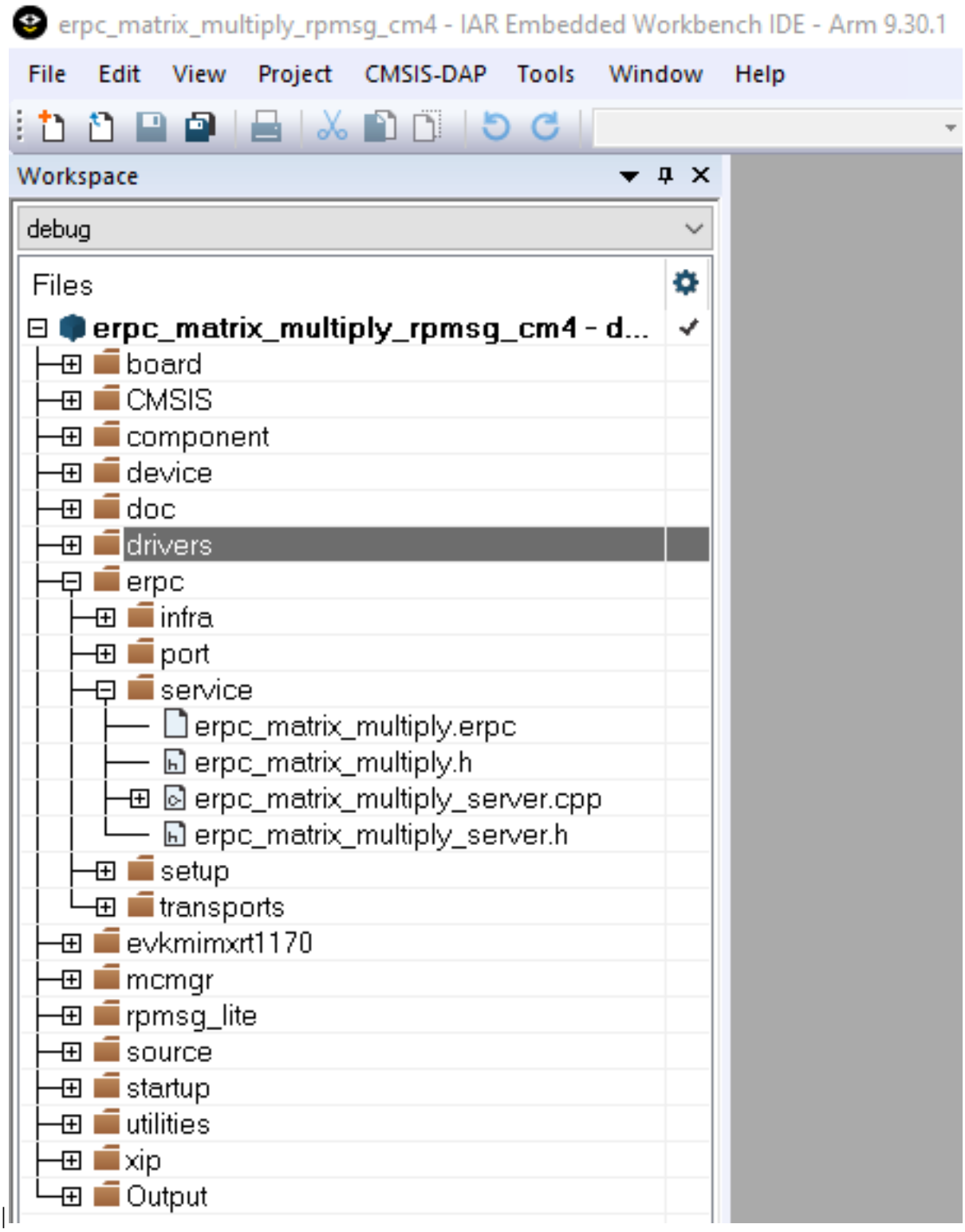
Server related generated files

The server-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/s



Parent topic: Multicore server application

Server infrastructure files

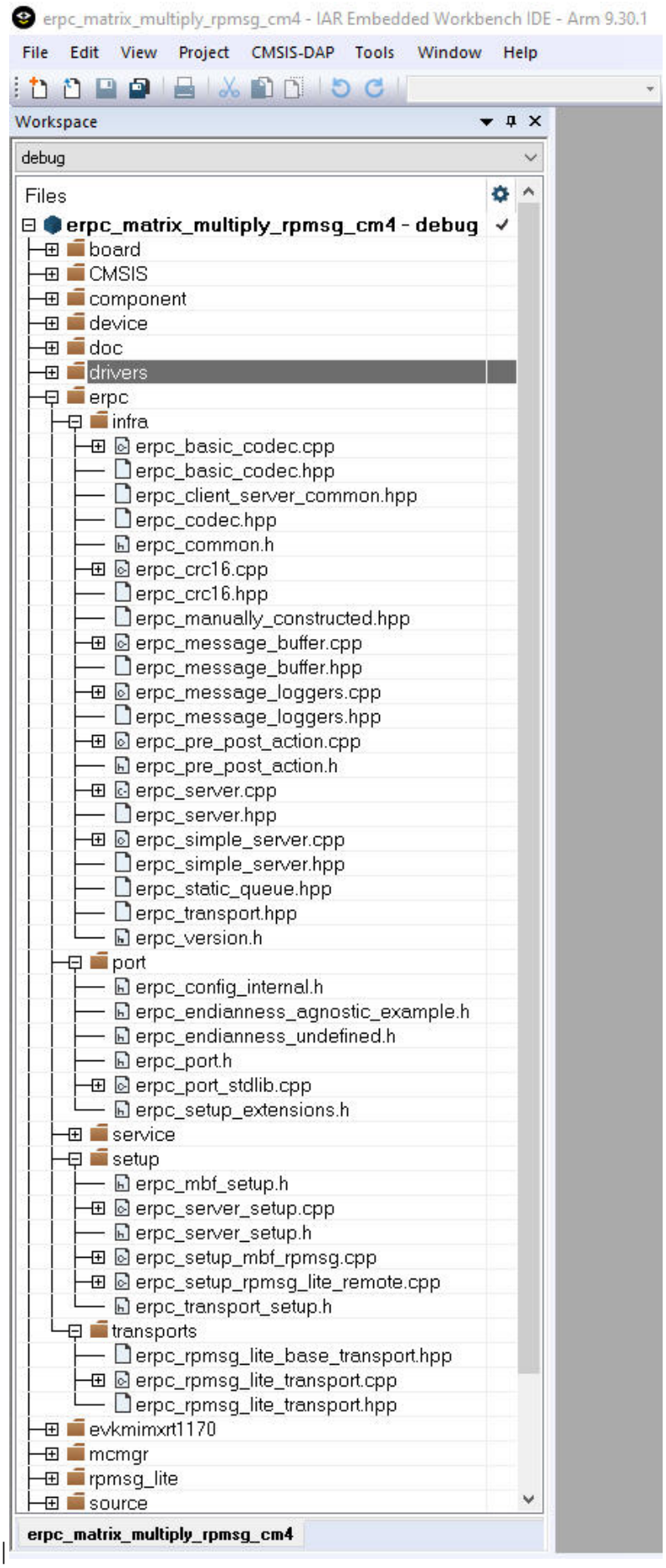
The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the *C/C++*

language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, erpc_server.hpp, erpc_server.cpp, erpc_simple_server.hpp, and erpc_simple_server.cpp, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The erpc_common.hpp file is used for common eRPC definitions, typedefs, and enums.
  - The erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: erpc_message_buffer.h and erpc_message_buffer.cpp.
  - The erpc_transport.h file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
  - erpc_port_stdlib.cpp file ensures adaptation to stdlib.
  - erpc_config_internal.h internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The erpc_server_setup.h and erpc_server_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
  - The erpc_transport_setup.h and erpc_setup_rpmsg_lite_remote.cpp files needs to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files need to be added into the server project.


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_server.cpp
erpc_server.hpp
erpc_simple_server.cpp
erpc_simple_server.hpp
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_mbf_setup.h
erpc_server_setup.cpp
erpc_server_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_remote.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

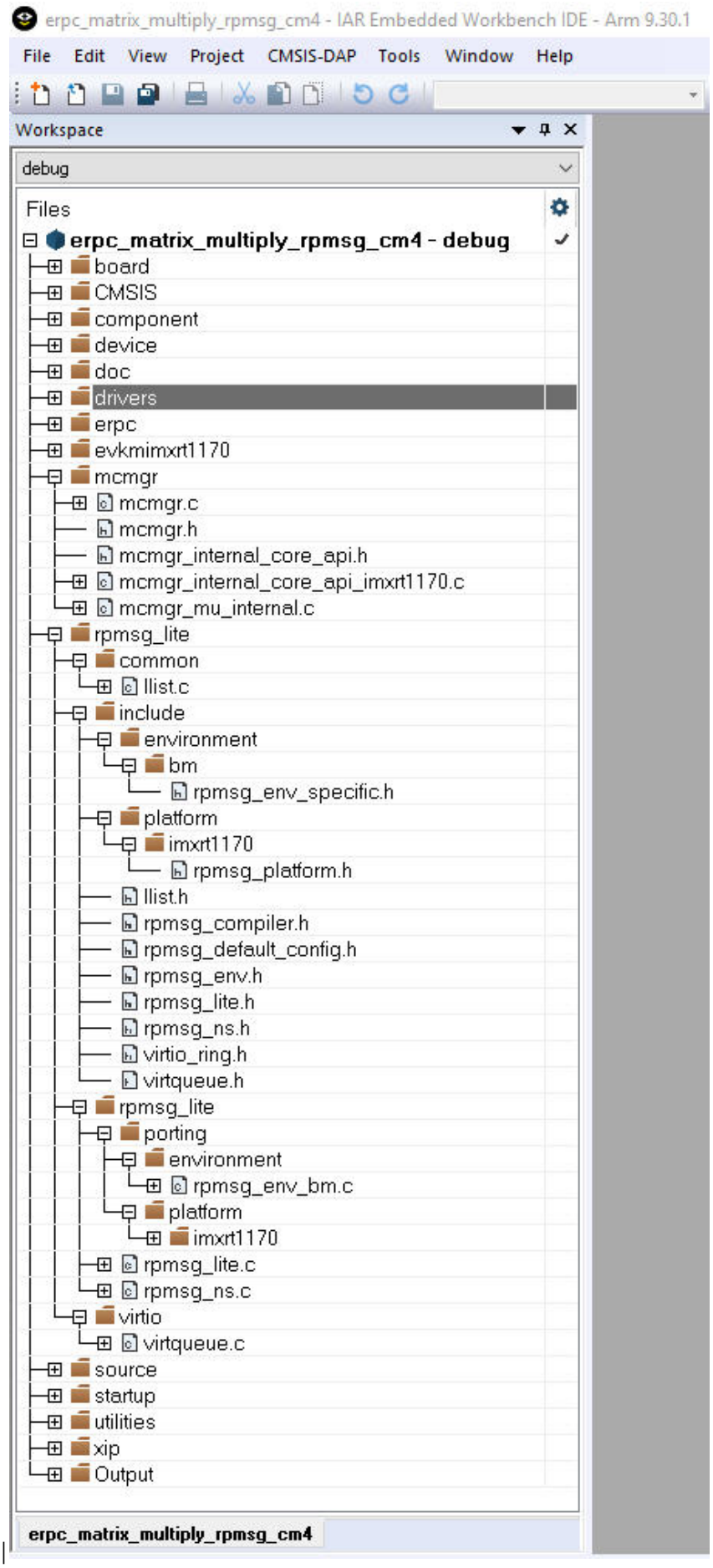
Server multicore infrastructure files

Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

Server user code

The server's user code is stored in the main_core1.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4

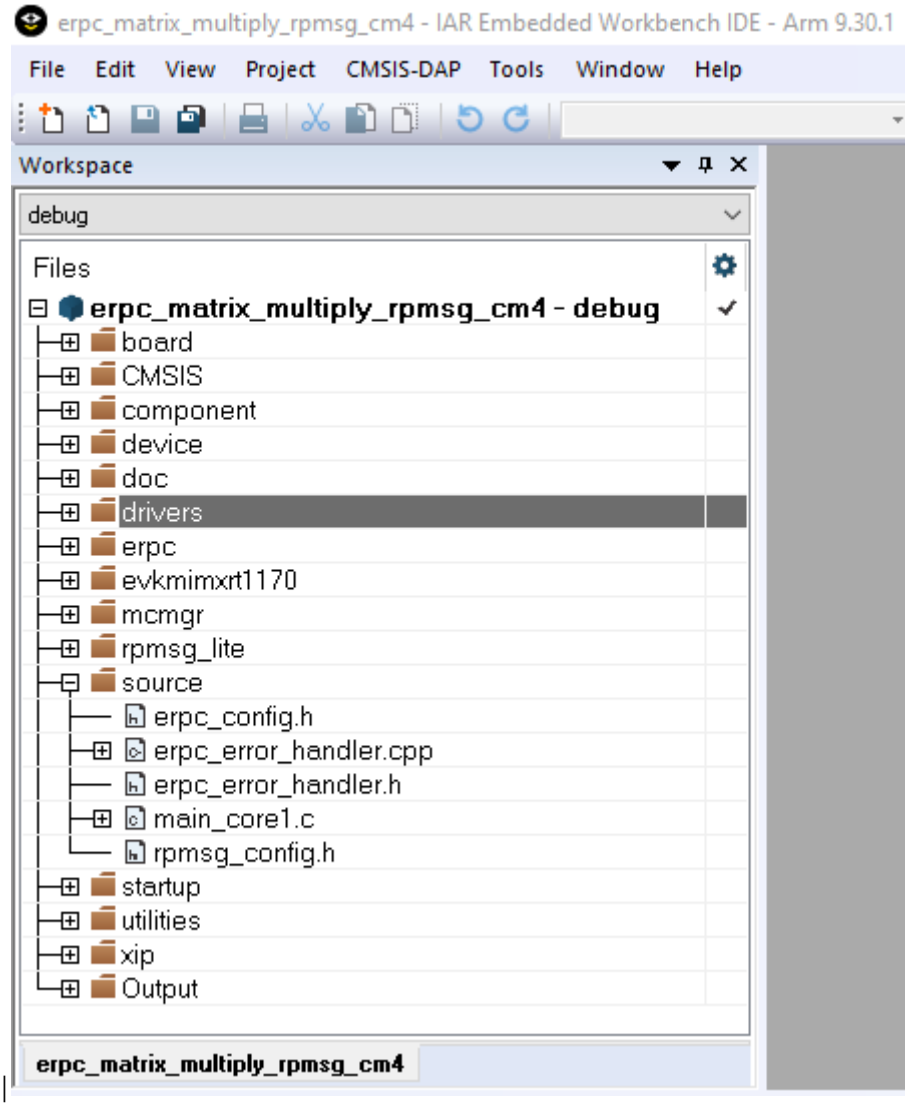
The main_core1.c file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the erpc_error_handler.h and erpc_error_handler.cpp files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
  ...
}
int main()
{
  ...
  /* RPMsg-Lite transport layer initialization */
  erpc_transport_t transport;
  transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
  ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_rpmsg_init(transport);
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server);
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg* folder.



Parent topic: Multicore server application

Parent topic:*Create an eRPC application*

Multicore client application

The “Matrix multiply” eRPC client project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar

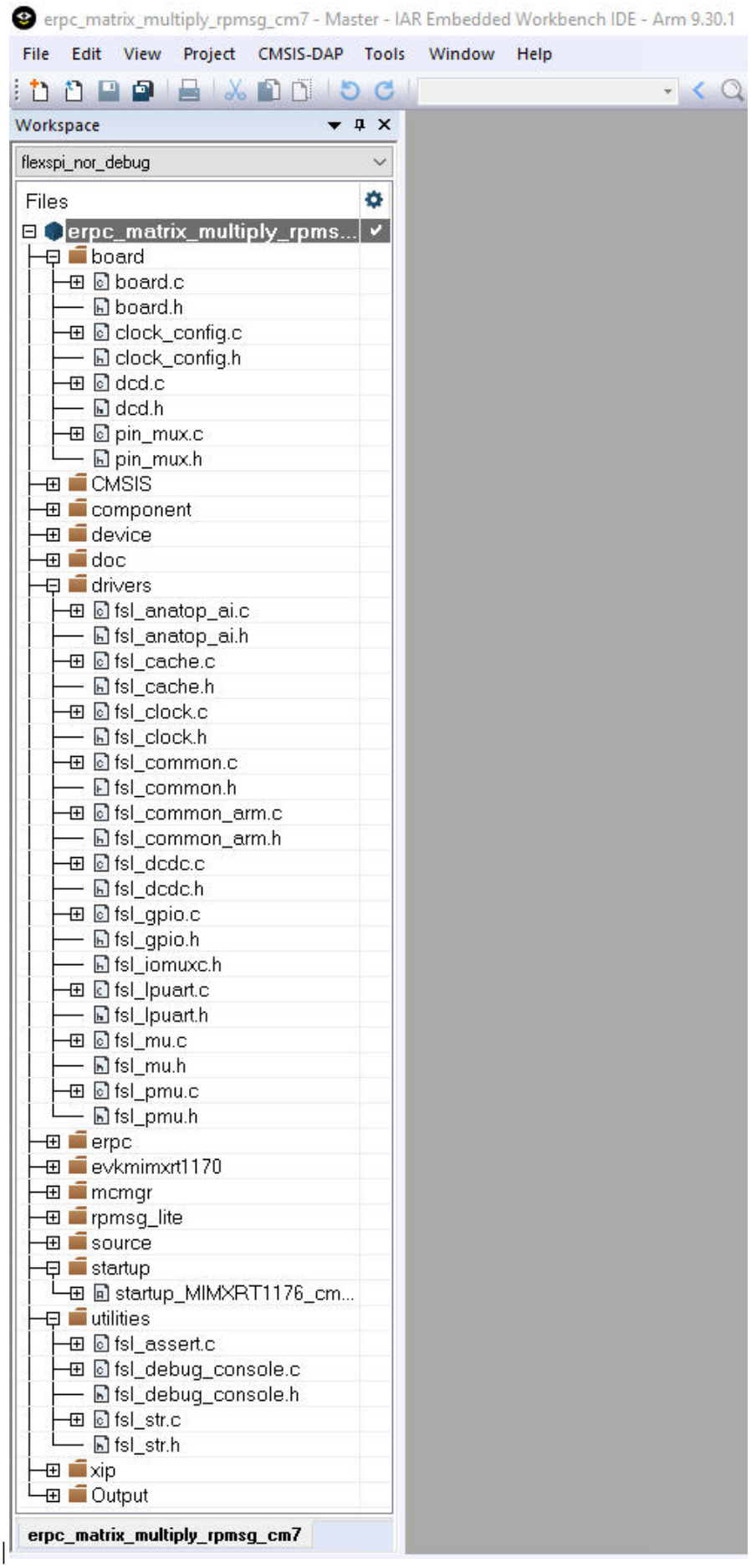
Project files for the eRPC client have the _cm7 suffix.

Client project basic source files

The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- *<MCUXpressoSDK_install_dir>/devices/<device>*

- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

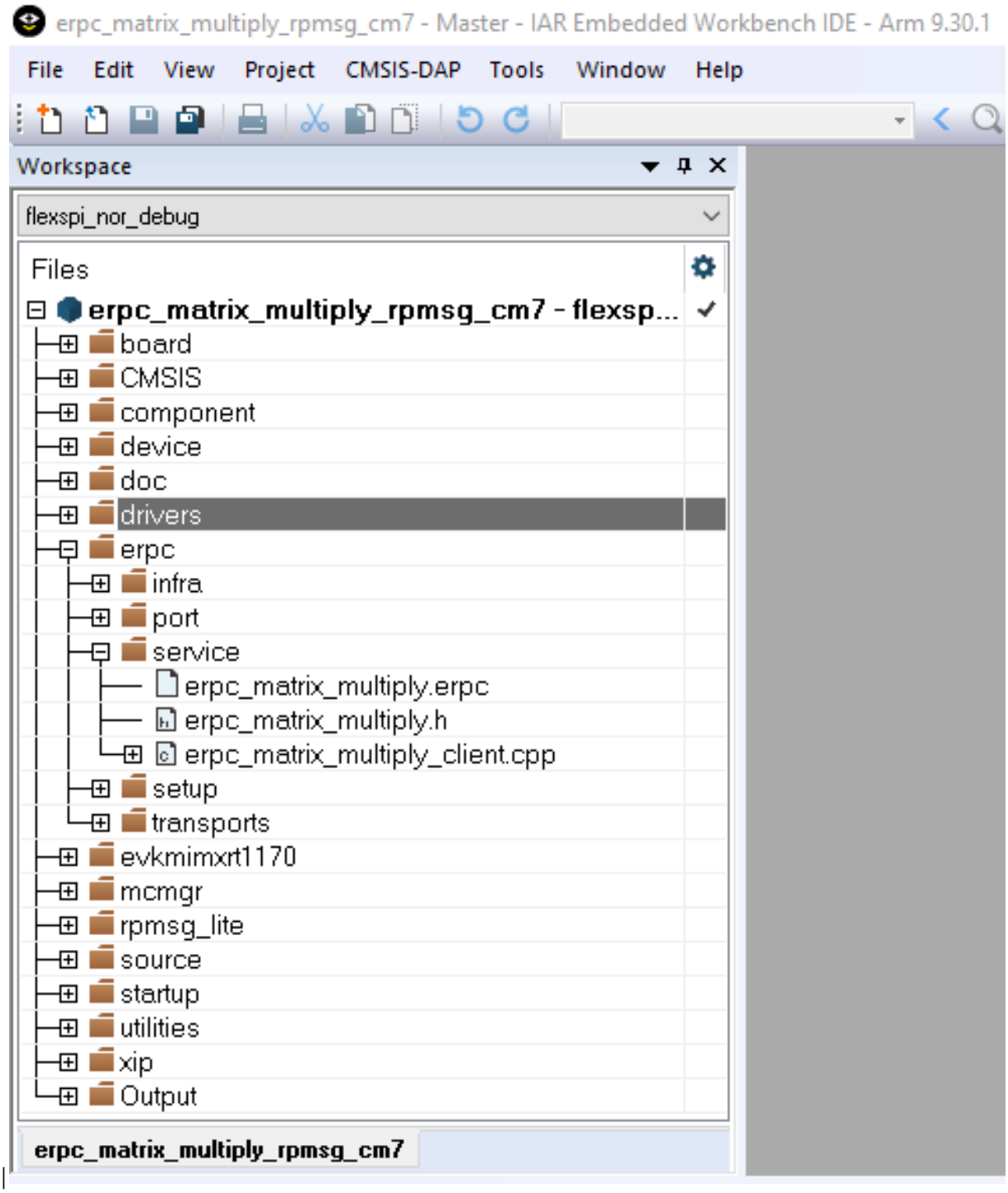
Parent topic: Multicore client application

Client-related generated files

The client-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/* folder.



Parent topic: Multicore client application

Client infrastructure files

The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, erpc_client_manager.h and erpc_client_manager.cpp, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- erpc_common.h file is used for common eRPC definitions, typedefs, and enums.
- erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: erpc_message_buffer.hpp and erpc_message_buffer.cpp.
- erpc_transport.hpp file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

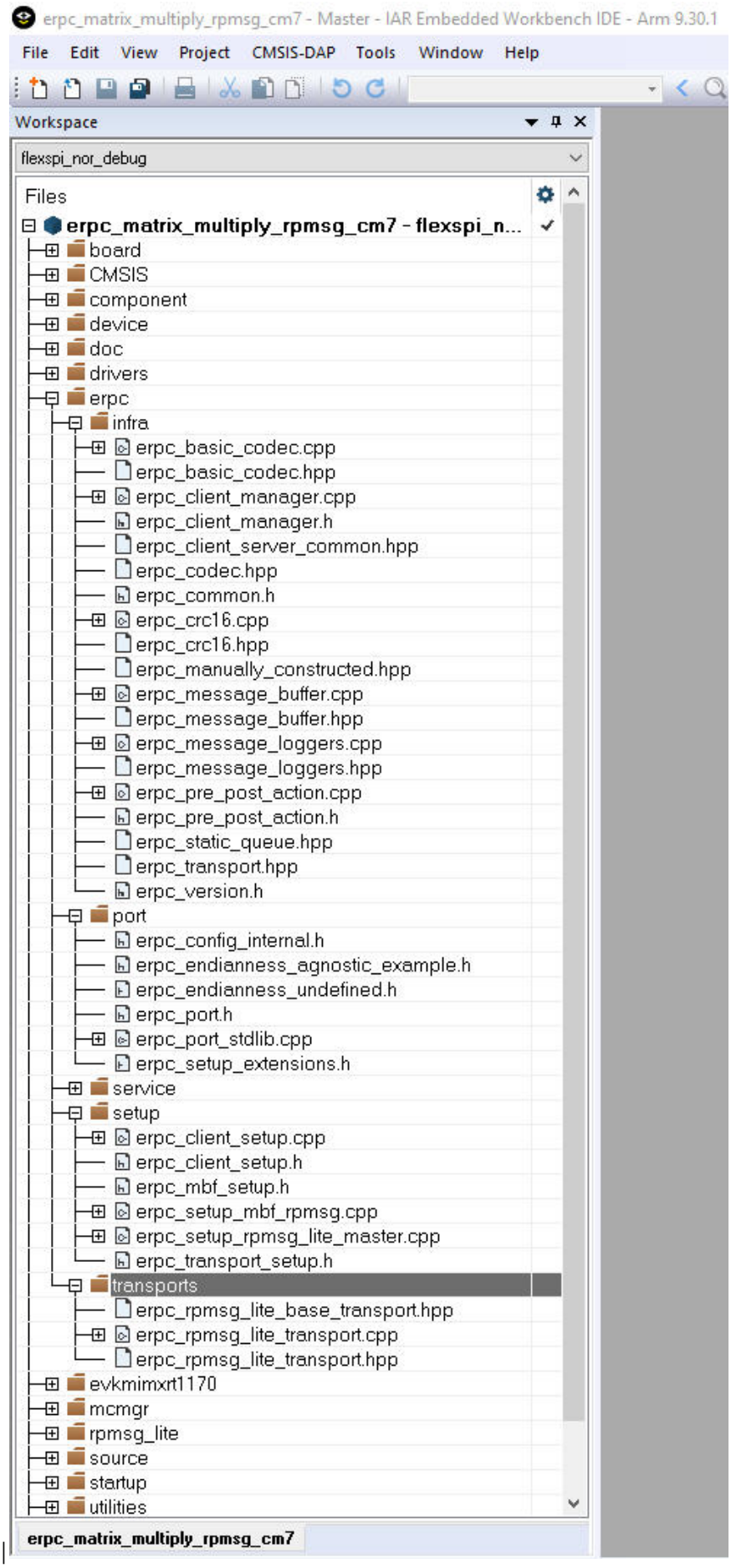
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- erpc_client_setup.h and erpc_client_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
- erpc_transport_setup.h and erpc_setup_rpmsg_lite_master.cpp files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files needs to be added into the client project.


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_n...
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_manager.cpp
erpc_client_manager.h
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_client_setup.cpp
erpc_client_setup.h
erpc_mbf_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_master.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
utilities
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

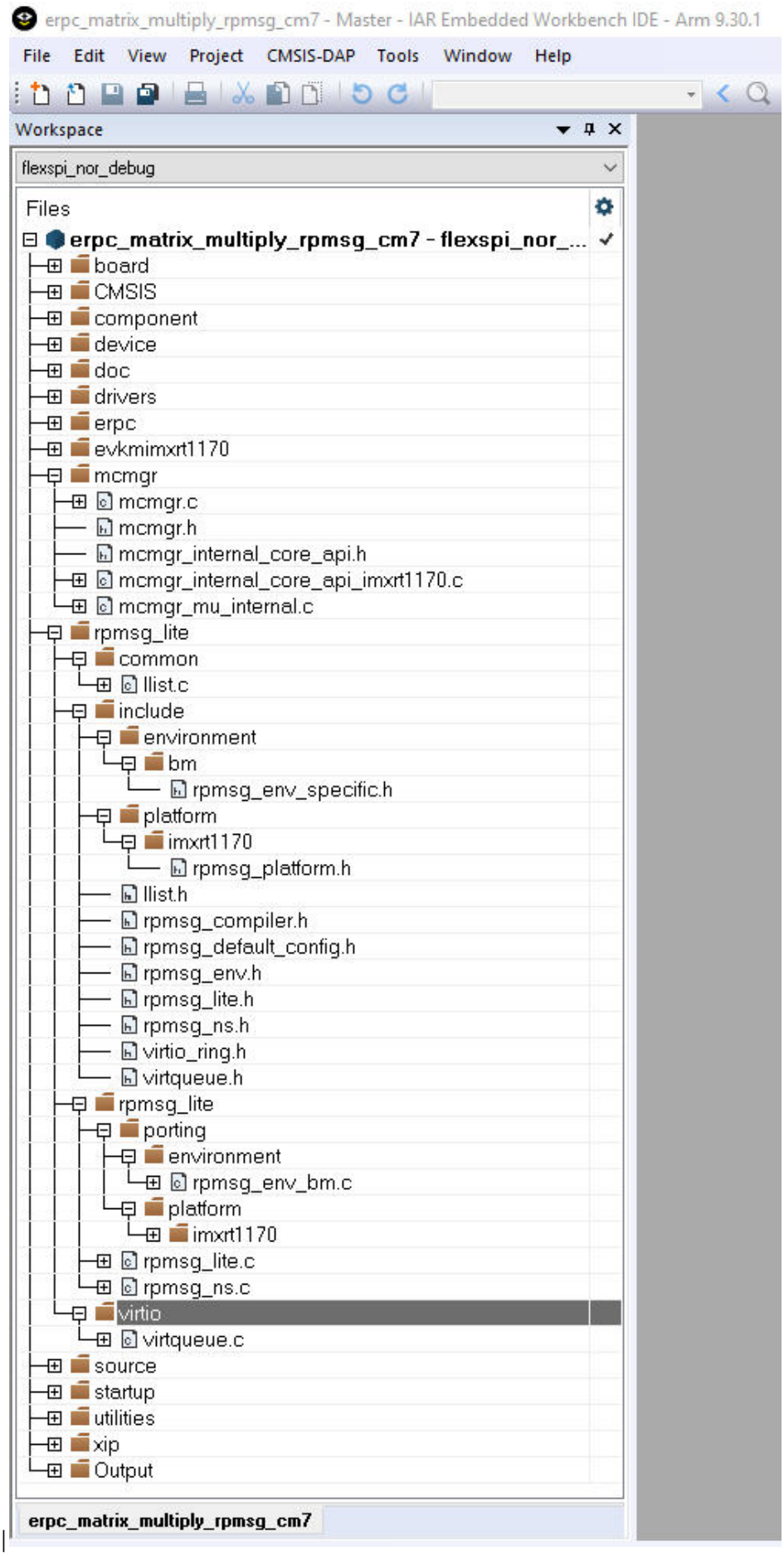
Client multicore infrastructure files

Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_nor_...
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

Client user code

The client's user code is stored in the main_core0.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_example/erpc_matrix_multiply_rpmsg/cm7

The main_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc_error_handler.h and erpc_error_handler.cpp files.

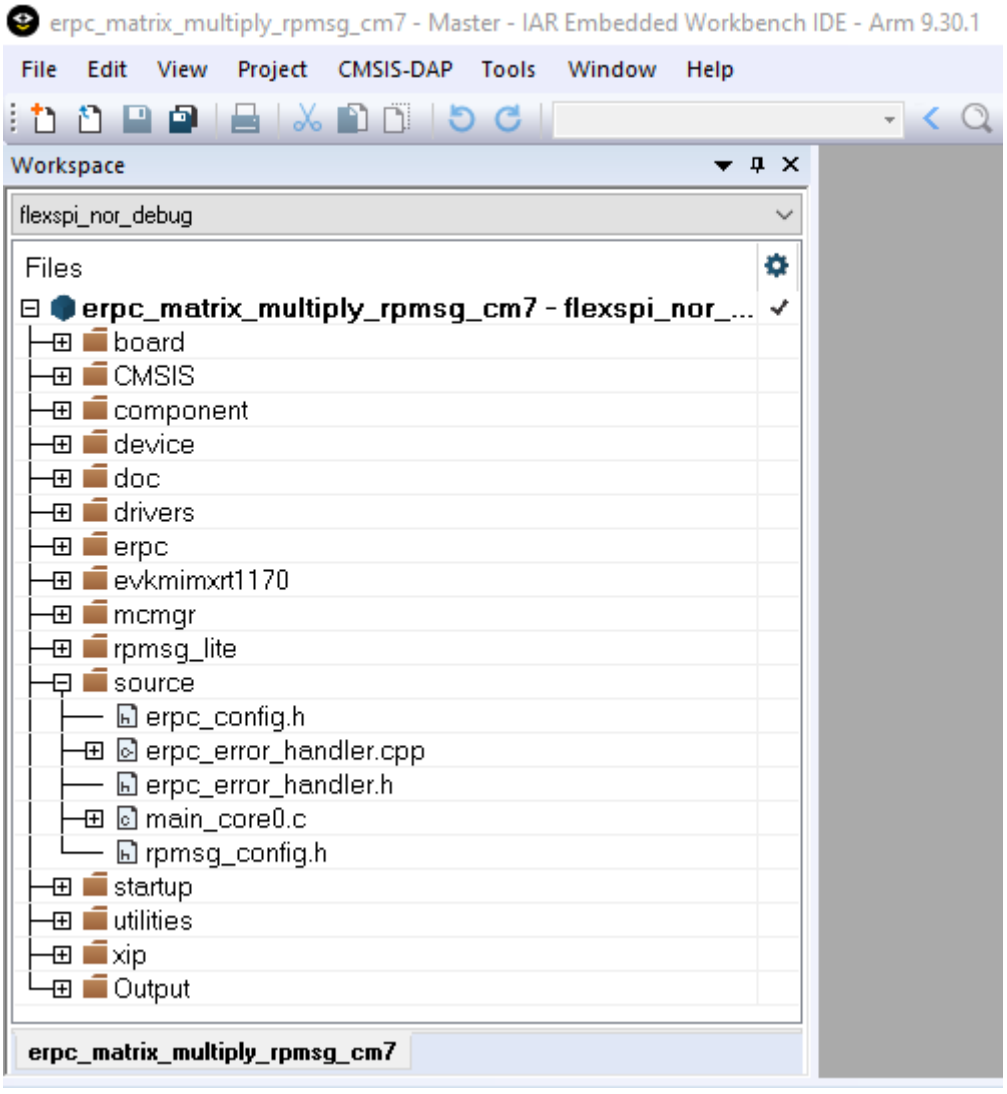
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg



|

|

Parent topic: Multicore client application

Parent topic:*Create an eRPC application*

Multiprocessor server application

The “Matrix multiply” eRPC server project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir»/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Server user code

The server's user code is stored in the main_server.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_lay* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
  ...
}
int main()
{
  ...
  /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
  erpc_transport_t transport;
  transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_dynamic_init();
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server)
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Parent topic: Multiprocessor server application

Multiprocessor client application

The "Matrix multiply" eRPC client project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<tr* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.hpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Client user code

The client's user code is stored in the main_client.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

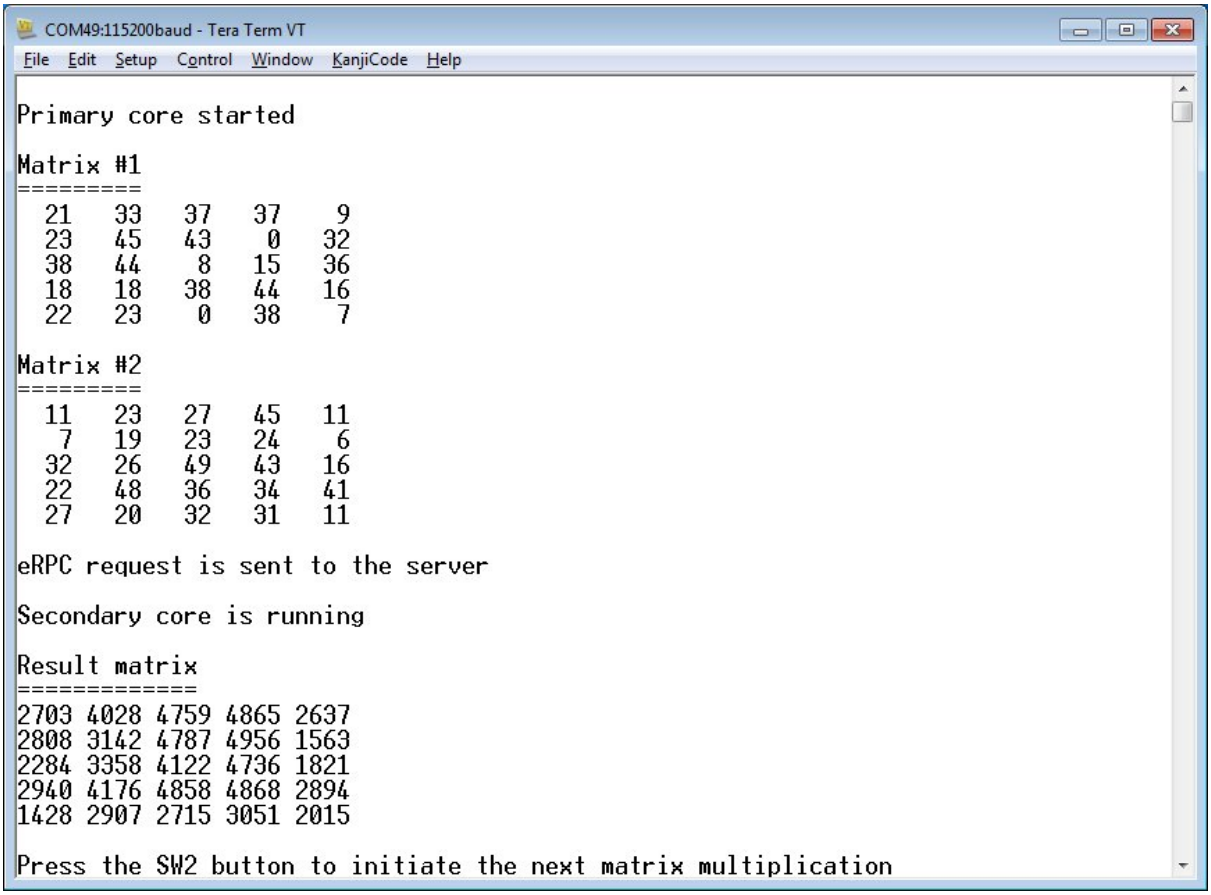
Parent topic: Multiprocessor client application

Parent topic: Multiprocessor server application

Parent topic:*Create an eRPC application*

Running the eRPC application

Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the <*MCUXpressoSDK_install_dir*>/*docs* folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:*Create an eRPC application*

Parent topic:*eRPC example*

Other uses for an eRPC implementation

The eRPC implementation is generic, and its use is not limited to just embedded applications. When creating an eRPC application outside the embedded world, the same principles apply. For example, this manual can be used to create an eRPC application for a PC running the Linux operating system. Based on the used type of transport medium, existing transport layers can be used, or new transport layers can be implemented.

For more information and erpc updates see the github.com/EmbeddedRPC.

Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Changelog eRPC

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

Unreleased

Added

- Added Zephyr IPC Service transport layer support.
- Zephyr examples added support for RT600 and RT700.

Updated

- eRPC Zephyr module port updated for Zephyr version 4.4

Fixed

- Python code of the eRPC infrastructure was updated to match the proper python code style, add type annotations and improve readability.
- eRPC: Several MISRA violations addressed.

1.14.0

Added

- Added Cmake/Kconfig support.
- Made java code jdk11 compliant, GitHub PR #432.
- Added imxrt1186 support into mu transport layer.
- erpcgen: Added assert for listType before usage, GitHub PR #406.

Fixed

- eRPC: Sources reformatted.
- erpc: Fixed typo in semaphore get (mutex -> semaphore), and write it can fail in case of timeout, GitHub PR #446.
- erpc: Free the arbitrated client token from client manager, GitHub PR #444.
- erpc: Fixed Makefile, install the erpc_simple_server header, GitHub PR #447.
- erpc_python: Fixed possible AttributeError and OSError on calling TCPTransport.close(), GitHub PR #438.
- Examples and tests consolidated.

1.13.0

Added

- erpc: Add BSD-3 license to endianness agnostic files, GitHub PR #417.
- eRPC: Add new Zephyr-related transports (zephyr_uart, zephyr_mbox).
- eRPC: Add new Zephyr-related examples.

Fixed

- eRPC,erpcgen: Fixing/improving markdown files, GitHub PR #395.
- eRPC: Fix Python client TCPTransports not being able to close, GitHub PR #390.
- eRPC,erpcgen: Align switch brackets, GitHub PR #396.
- erpc: Fix zephyr uart transport, GitHub PR #410.
- erpc: UART ZEPHYR Transport stop to work after a few transactions when using USB-CDC resolved, GitHub PR #420.

Removed

- eRPC,erpcgen: Remove cstbool library, GitHub PR #403.

1.12.0

Added

- eRPC: Add dynamic/static option for transport init, GitHub PR #361.
- eRPC,erpcgen: Winsock2 support, GitHub PR #365.
- eRPC,erpcgen: Feature/support multiple clients, GitHub PR #271.
- eRPC,erpcgen: Feature/buffer head - Framed transport header data stored in MessageBuffer, GitHub PR #378.
- eRPC,erpcgen: Add experimental Java support.

Fixed

- eRPC: Fix receive error value for spidev, GitHub PR #363.
- eRPC: UartTransport::init adaptation to changed driver.
- eRPC: Fix typo in assert, GitHub PR #371.
- eRPC,erpcgen: Move enums to enum classes, GitHub PR #379.
- eRPC: Fixed rpmsg tty transport to work with serial transport, GitHub PR #373.

1.11.0

Fixed

- eRPC: Makefiles update, GitHub PR #301.
- eRPC: Resolving warnings in Python, GitHub PR #325.
- eRPC: Python3.8 is not ready for usage of typing.Any type, GitHub PR #325.
- eRPC: Improved codec function to use reference instead of address, GitHub PR #324.
- eRPC: Fix NULL check for pending client creation, GitHub PR #341.
- eRPC: Replace sprintf with snprintf, GitHub PR #343.
- eRPC: Use MU_SendMsg blocking call in MU transport.
- eRPC: New LPSPI and LPI2C transport layers.
- eRPC: Freeing static objects, GitHub PR #353.
- eRPC: Fixed casting in deinit functions, GitHub PR #354.
- eRPC: Align LIBUSBSIO.GetNumPorts API use with libusbsio python module v. 2.1.11.
- erpcgen: Renamed temp variable to more generic one, GitHub PR #321.
- erpcgen: Add check that string read is not more than max length, GitHub PR #328.
- erpcgen: Move to g++ in pytest, GitHub PR #335.
- erpcgen: Use build=release for make, GitHub PR #334.
- erpcgen: Removed boost dependency, GitHub PR #346.
- erpcgen: Mingw support, GitHub PR #344.
- erpcgen: VS build update, GitHub PR #347.
- erpcgen: Modified name for common types macro scope, GitHub PR #337.
- erpcgen: Fixed memcpy for template, GitHub PR #352.
- eRPC,erpcgen: Change default build target to release + adding artefacts, GitHub PR #334.
- eRPC,erpcgen: Remove redundant includes, GitHub PR #338.
- eRPC,erpcgen: Many minor code improvements, GitHub PR #323.

1.10.0

Fixed

- eRPC: MU transport layer switched to blocking MU_SendMsg() API use.

1.10.0

Added

- eRPC: Add TCP_NODELAY option to python, GitHub PR #298.

Fixed

- eRPC: MUTransport adaptation to new supported SoCs.
- eRPC: Simplifying CI with installing dependencies using shell script, GitHub PR #267.
- eRPC: Using event for waiting for sock connection in TCP python server, formatting python code, C specific includes, GitHub PR #269.

- eRPC: Endianness agnostic update, GitHub PR #276.
- eRPC: Assertion added for functions which are returning status on freeing memory, GitHub PR #277.
- eRPC: Fixed closing arbitrator server in unit tests, GitHub PR #293.
- eRPC: Makefile updated to reflect the correct header names, GitHub PR #295.
- eRPC: Compare value length to used length() in reading data from message buffer, GitHub PR #297.
- eRPC: Replace EXPECT_TRUE with EXPECT_EQ in unit tests, GitHub PR #318.
- eRPC: Adapt rpmsg_lite based transports to changed rpmsg_lite_wait_for_link_up() API parameters.
- eRPC, erpcgen: Better distuingish which file can and cannot by linked by C linker, GitHub PR #266.
- eRPC, erpcgen: Stop checking if pointer is NULL before sending it to the erpc_free function, GitHub PR #275.
- eRPC, erpcgen: Changed api to count with more interfaces, GitHub PR #304.
- erpcgen: Check before reading from heap the buffer boundaries, GitHub PR #287.
- erpcgen: Several fixes for tests and CI, GitHub PR #289.
- erpcgen: Refactoring erpcgen code, GitHub PR #302.
- erpcgen: Fixed assigning const value to enum, GitHub PR #309.
- erpcgen: Enable runTesttest_enumErrorCode_allDirection, serialize enums as int32 instead of uint32.

1.9.1

Fixed

- eRPC: Construct the USB CDC transport, rather than a client, GitHub PR #220.
- eRPC: Fix premature import of package, causing failure when attempting installation of Python library in a clean environment, GitHub PR #38, #226.
- eRPC: Improve python detection in make, GitHub PR #225.
- eRPC: Fix several warnings with deprecated call in pytest, GitHub PR #227.
- eRPC: Fix freeing union members when only default need be freed, GitHub PR #228.
- eRPC: Fix making test under Linux, GitHub PR #229.
- eRPC: Assert costumizing, GitHub PR #148.
- eRPC: Fix corrupt clientList bug in TransportArbitrator, GitHub PR #199.
- eRPC: Fix build issue when invoking g++ with -Wno-error=free-nonheap-object, GitHub PR #233.
- eRPC: Fix inout cases, GitHub PR #237.
- eRPC: Remove ERPC_PRE_POST_ACTION dependency on return type, GitHub PR #238.
- eRPC: Adding NULL to ptr when codec function failed, fixing memcpy when fail is present during deserialization, GitHub PR #253.
- eRPC: MessageBuffer usage improvement, GitHub PR #258.
- eRPC: Get rid for serial and enum34 dependency (enum34 is in python3 since 3.4 (from 2014)), GitHub PR #247.

- eRPC: Several MISRA violations addressed.
- eRPC: Fix timeout for Freertos semaphore, GitHub PR #251.
- eRPC: Use of rpmsg_lite_wait_for_link_up() in rpmsg_lite based transports, GitHub PR #223.
- eRPC: Fix codec nullptr dereferencing, GitHub PR #264.
- erpcgen: Fix two syntax errors in erpcgen Python output related to non-encapsulated unions, improved test for union, GitHub PR #206, #224.
- erpcgen: Fix serialization of list/binary types, GitHub PR #240.
- erpcgen: Fix empty list parsing, GitHub PR #72.
- erpcgen: Fix templates for malloc errors, GitHub PR #110.
- erpcgen: Get rid of encapsulated union declarations in global scale, improve enum usage in unions, GitHub PR #249, #250.
- erpcgen: Fix compile error:UniqueIdChecker.cpp:156:104:'sort' was not declared, GitHub PR #265.

1.9.0

Added

- eRPC: Allow used LIBUSBSIO device index being specified from the Python command line argument.

Fixed

- eRPC: Improving template usage, GitHub PR #153.
- eRPC: run_clang_format.py cleanup, GitHub PR #177.
- eRPC: Build TCP transport setup code into liberpc, GitHub PR #179.
- eRPC: Fix multiple definitions of g_client error, GitHub PR #180.
- eRPC: Fix memset past end of buffer in erpc_setup_mbf_static.cpp, GitHub PR #184.
- eRPC: Fix deprecated error with newer pytest version, GitHub PR #203.
- eRPC, erpcgen: Static allocation support and usage of rpmsg static FreeRTOSs related APi, GitHub PR #168, #169.
- erpcgen: Remove redundant module imports in erpcgen, GitHub PR #196.

1.8.1

Added

- eRPC: New i2c_slave_transport trasnport introduced.

Fixed

- eRPC: Fix misra erpc c, GitHub PR #158.
- eRPC: Allow conditional compilation of message_loggers and pre_post_action.
- eRPC: (D)SPI slave transports updated to avoid busy loops in rtos environments.
- erpcgen: Re-implement EnumMember::hasValue(), GitHub PR #159.
- erpcgen: Fixing several misra issues in shim code, erpcgen and unit tests updated, GitHub PR #156.

- erpcgen: Fix bison file, GitHub PR #156.

1.8.0

Added

- eRPC: Support win32 thread, GitHub PR #108.
- eRPC: Add mbed support for malloc() and free(), GitHub PR #92.
- eRPC: Introduced pre and post callbacks for eRPC call, GitHub PR #131.
- eRPC: Introduced new USB CDC transport.
- eRPC: Introduced new Linux spidev-based transport.
- eRPC: Added formatting extension for VSC, GitHub PR #134.
- erpcgen: Introduce ustring type for unsigned char and force cast to char*, GitHub PR #125.

Fixed

- eRPC: Update makefile.
- eRPC: Fixed warnings and error with using MessageLoggers, GitHub PR #127.
- eRPC: Extend error msg for python server service handle function, GitHub PR #132.
- eRPC: Update CMSIS UART transport layer to avoid busy loops in rtos environments, introduce semaphores.
- eRPC: SPI transport update to allow usage without handshaking GPIO.
- eRPC: Native _WIN32 erpc serial transport and threading.
- eRPC: Arbitrator deadlock fix, TCP transport updated, TCP setup functions introduced, GitHub PR #121.
- eRPC: Update of matrix_multiply.py example: Add –serial and –baud argument, GitHub PR #137.
- eRPC: Update of .clang-format, GitHub PR #140.
- eRPC: Update of erpc_framed_transport.cpp: return error if received message has zero length, GitHub PR #141.
- eRPC, erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136, #139.
- eRPC, erpcgen: Core re-formatted using Clang version 10.
- erpcgen: Enable deallocation in server shim code when callback/function pointer used as out parameter in IDL.
- erpcgen: Removed '$' character from generated symbol name in '_$union' suffix, GitHub PR #103.
- erpcgen: Resolved mismatch between C++ and Python for callback index type, GitHub PR #111.
- erpcgen: Python generator improvements, GitHub PR #100, #118.
- erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136.

1.7.4

Added

- eRPC: Support MU transport unit testing.
- eRPC: Adding mbed os support.

Fixed

- eRPC: Unit test code updated to handle service add and remove operations.
- eRPC: Several MISRA issues in rpmsg-based transports addressed.
- eRPC: Fixed Linux/TCP acceptance tests in release target.
- eRPC: Minor documentation updates, code formatting.
- erpcgen: Whitespace removed from C common header template.

1.7.3

Fixed

- eRPC: Improved the test_callbacks logic to be more understandable and to allow requested callback execution on the server side.
- eRPC: TransportArbitrator::prepareClientReceive modified to avoid incorrect return value type.
- eRPC: The ClientManager and the ArbitratedClientManager updated to avoid performing client requests when the previous serialization phase fails.
- erpcgen: Generate the shim code for destroy of statically allocated services.

1.7.2

Added

- eRPC: Add missing doxygen comments for transports.

Fixed

- eRPC: Improved support of const types.
- eRPC: Fixed Mac build.
- eRPC: Fixed serializing python list.
- eRPC: Documentation update.

1.7.1

Fixed

- eRPC: Fixed semaphore in static message buffer factory.
- erpcgen: Fixed MU received error flag.
- erpcgen: Fixed tcp transport.

1.7.0

Added

- eRPC: List names are based on their types. Names are more deterministic.
- eRPC: Service objects are as a default created as global static objects.

- eRPC: Added missing doxygen comments.
- eRPC: Added support for 64bit numbers.
- eRPC: Added support of program language specific annotations.

Fixed

- eRPC: Improved code size of generated code.
- eRPC: Generating crc value is optional.
- eRPC: Fixed CMSIS Uart driver. Removed dependency on KSDK.
- eRPC: Forbid users use reserved words.
- eRPC: Removed outByref for function parameters.
- eRPC: Optimized code style of callback functions.

1.6.0

Added

- eRPC: Added @nullable support for scalar types.

Fixed

- eRPC: Improved code size of generated code.
- eRPC: Improved eRPC nested calls.
- eRPC: Improved eRPC list length variable serialization.

1.5.0

Added

- eRPC: Added support for unions type non-wrapped by structure.
- eRPC: Added callbacks support.
- eRPC: Added support @external annotation for functions.
- eRPC: Added support @name annotation.
- eRPC: Added Messaging Unit transport layer.
- eRPC: Added RPMSG Lite RTOS TTY transport layer.
- eRPC: Added version verification and IDL version verification between eRPC code and eRPC generated shim code.
- eRPC: Added support of shared memory pointer.
- eRPC: Added annotation to forbid generating const keyword for function parameters.
- eRPC: Added python matrix multiply example.
- eRPC: Added nested call support.
- eRPC: Added struct member “byref” option support.
- eRPC: Added support of forward declarations of structures
- eRPC: Added Python RPMsg Multiendpoint kernel module support
- eRPC: Added eRPC sniffer tool

1.4.0

Added

- eRPC: New RPMsg-Lite Zero Copy (RPMsgZC) transport layer.

Fixed

- eRPC: win_flex_bison.zip for windows updated.
- eRPC: Use one codec (instead of inCodec outCodec).

[1.3.0]

Added

- eRPC: New annotation types introduced (@length, @max_length, ...).
- eRPC: Support for running both erpc client and erpc server on one side.
- eRPC: New transport layers for (LP)UART, (D)SPI.
- eRPC: Error handling support.

[1.2.0]

Added

- eRPC source directory organization changed.
- Many eRPC improvements.

[1.1.0]

Added

- Multicore SDK 1.1.0 ported to KSDK 2.0.0.

[1.0.0]

Added

- Initial Release

3.7 Multimedia

3.7.1 VGLite Graphics Driver

IMXRTVGLITEAPIRM

Introduction The VGLite Graphics API (Application Programming Interface) is designed to support 2D vector and 2D raster-based operations for rendering the interactive user interface that may include menus, fonts, curves, and images. The goal is to provide the maximum 2D vector/raster rendering performance, while keeping the memory footprint to the minimum.

Note: This document contains proprietary information of VeriSilicon Holdings Co., Ltd, and Vivante Corporation.

VGLite Graphics API

The Vivante VGLite Graphics API is used to control the Vivante vector graphics hardware units that provide accelerated vector and raster operations.

The Vivante VGLite API is developed for use with Vivante GCNanoLiteV, GCNanoUltraV, GCNanoV, GC355, and GC555 hardware. GC355 and GC555 support the Khronos OpenVG 1.1 feature set, while GCNanoLiteV, GCNanoUltraV and GCNanoV have a feature set smaller than that required to pass Khronos OpenVG CTS.

The VGLite API driver V4 is a new design and implementation of the driver (from 2023Q1) to support the new generation 2.5D GPU (GC555), and the previous 2.5D GPU releases (GC255, GC265, GC355). The new V4 driver supports the new and improved VGLite API (version 3.0) and can generate the most CPU-efficient, customized driver build for a specific 2.5D GPU release based on the hardware feature set.

VGLite API supported features include: Porter-Duff Blending, Gradient Controls, Fast Clear, Arbitrary Rotations, Path Filling rules, Path painting, and Pattern Path Filling.

By default, VGLite API driver V4 supports one implicit global application context in a single thread. VGLite V4 driver does not support multithreaded applications, which is not suitable for embedded IoT devices.

Parent topic:*Introduction*

API function group

The VGLite Graphics API has been designed to have independent function groups. It is permissible for a user to use only one of the function groups in the VGLite application:

- **Initialization** is used for initializing hardware and software structures
- **Blit API** is used for the raster part of rendering
- **Draw API** is used for 2D vector-based draw operations

Parent topic:*Introduction*

API files

The VGLite source code is available as part of the NXP MCUXpresso SDK:

The VGLite graphics API functions are defined in the header file VGLite/inc/vg_lite.h.

All VGLite enumerations and data types are defined in VGLite/inc/vg_lite.h.

Parent topic:*Introduction*

Hardware versions

The Vivante VGLite API is compatible with a range of Vivante Vector Graphics IPs including: GCNanoLiteV, GCNanoUltraV, GCNanoV, GC355, and GC555.

Note: A specific hardware version has customized feature set that may limit hardware support for some VGLite API options. The VGLite application can use the vg_lite_query_feature API to query specific VGLite feature availability.

Users can also check the VGLite/VGLite/vg_lite_options.h file which includes CHIPID, REVISION, CID to identify specific HW releases, and the gcFEATURE_VG_* macros to define the feature set for the HW release.

The gcFEATURE_VG_* macro values (except a few SW features) should NOT be changed. Otherwise, the VGLite driver does not function correctly on the specific HW release. Users can change the “SW Features” macro values to disable some software features, unnecessary error checks, or enable VGLite API trace for debug purposes.

.

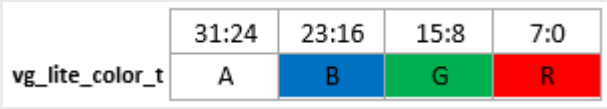
Parent topic:*Introduction*

Common parameters and error values This chapter provides an overview of the common parameter types and the enumeration used for error reporting.

Common parameter types

The VGLite graphics API uses a naming convention scheme wherein definitions are preceded by vg_lite.

Below is the list of types and structures in the driver implementation.

Nam	Type-def	Value
vg_li	int	A signed 32-bit integer 0: FALSE; 1: TRUE.
vg_li	char	A signed 8-bit integer
vg_li	un-signe char	An unsigned 8-bit integer
vg_li	short	A signed 16-bit integer
vg_li	un-signe short	An unsigned 16-bit integer
vg_li	int	A signed 32-bit integer
vg_li	un-signe int	An unsigned 32-bit integer
vg_li	un-signe long long	An unsigned 64-bit integer
vg_li	float	A 32-bit single precision floating point number
vg_li	dou-ble	A 64-bit double precision floating point number
vg_li	char	A signed 8-bit integer
vg_li	char*	A pointer to a character string
vg_li	void*	A generic address pointer (void *). On 32-bit OS, it is a 32-bit address pointer. On 64-bit OS, it is a 64-bit address pointer.
vg_li	void	The void type
vg_li	vg_lit	A 32-bit color value The color value specifies the color used in various functions. The color is formed using 8-bit RGBA channels. The red channel is in the lower 8-bit of the color value, followed by the green and blue channels. The alpha channel is in the upper 8-bit of the color value.  For L8 target formats, the RGB color is converted to L8 by using the default ITU-R BT.709 conversion rules.
VG_l	enum vg_lit	A signed 8-bit integer coordinate
VG_l	enum vg_lit	A signed 16 bit integer coordinate
VG_l	enum vg_lit	A signed 32-bit integer coordinate
VG_l	enum vg_lit	A 32-bit floating point coordinate

Parent topic:*Common parameters and error values*

Enumerations for error reporting

This section describes enumerations used for error reporting.

vg_lite_error_t enumeration

Most functions in the API include an error status via the vg_lite_error_tenumeration. API functions return the status of the command and will report VG_LITE_SUCCESS if successful with no errors. Possible error values include the values in the table below. vg_lite_error_tenumeration is used in many functions, including initialization, flush, blit, draw, gradient, and pattern functions.

vg_lite_error_t string values	Description
VG_LITE_GENERIC_IO	Cannot communicate with the kernel driver
VG_LITE_INVALID_ARGUMEN	An invalid argument was specified
VG_LITE_MULTI_THREAD_FA	Multi-thread/tasks fail *(available from June 2020)*
VG_LITE_NO_CONTEXT	No context specified
VG_LITE_NOT_SUPPORT	Function call is not supported. Hardware support is not available.
VG_LITE_OUT_OF_MEMORY	Out of memory (driver heap)
VG_LITE_OUT_OF_RESOURC	Out of resources (OS heap)
VG_LITE_SUCCESS	Successful with no errors
VG_LITE_TIMEOUT	Timeout
VG_LITE_ALREADY_EXISTS	Object exists *(available from August 2021)*
VG_LITE_NOT_ALIGNED	Data alignment error *(available from August 2021)*

Parent topic: Enumerations for error reporting

Parent topic:*Common parameters and error values*

Hardware product and feature information These query functions can be used to identify the product and its key features and to get VGLite driver information.

Enumerations for product and feature queries

This section describes enumerations used for product and feature queries.

vg_lite_feature_t enumeration

The following feature values may be queried for availability in compatible hardware. *(expanded March 2023 to support additional hardware for driver V4)*

Used in information function: vg_lite_query_feature.

vg_lite_feature_t string values	Description
gcFEATURE_BIT_VG_16PIXELS_ALIGN	Require 16 pixels aligned for the input pixel buffer
gcFEATURE_BIT_VG_24BIT	RGB888 or RGBA5658 formats support
gcFEATURE_BIT_VG_24BIT_PLANAR	24-bit planar format support
gcFEATURE_BIT_VG_AYUV_INPUT	AYUV input format support
gcFEATURE_BIT_VG_BORDER_CULLING	Border culling support
gcFEATURE_BIT_VG_COLOR_KEY	Color key support.
gcFEATURE_BIT_VG_COLOR_TRANSFORMATION	Color transform support.
gcFEATURE_BIT_VG_DEC_COMPRESS	DEC compression format output support

continues on next page

Table 48 – continued from previous page

vg_lite_feature_t string values	Description
gcFEATURE_BIT_VG_DITHER	Dither support
gcFEATURE_BIT_VG_DOUBLE_IMAGE	Support two image source inputs
gcFEATURE_BIT_VG_FLEXA	FLEXA interface support
gcFEATURE_BIT_VG_GAMMA	Gamma support
gcFEATURE_BIT_VG_GAUSSIAN_BLUR	Gaussian blur sampling support
gcFEATURE_BIT_VG_GLOBAL_ALPHA	Global alpha support
gcFEATURE_BIT_VG_HW_PREMULTIPLY	HW supports alpha premultiply for image
gcFEATURE_BIT_VG_IM_DEC_INPUT	DEC compressed format input support
gcFEATURE_BIT_VG_IM_FASTCLEAR	Fast Clear support
gcFEATURE_BIT_VG_IM_INDEX_FORMAT	Index format support for image
gcFEATURE_BIT_VG_IM_INPUT	Blit and draw API support
gcFEATURE_BIT_VG_IM_REPEAT_REFLECT	Image repeat reflect mode support
gcFEATURE_BIT_VG_INDEX_ENDIAN	Index format endian support
gcFEATURE_BIT_VG_LINEAR_GRADIENT_EXT	Support for extended linear gradient capabilities
gcFEATURE_BIT_VG_LVGL_SUPPORT	LVGL blend mode support
gcFEATURE_BIT_VG_MASK	Mask support
gcFEATURE_BIT_VG_MIRROR	Mirror support
gcFEATURE_BIT_VG_NEW_BLEND_MODE	New blend mode DARKEN/LIGHTEN support
gcFEATURE_BIT_VG_NEW_IMAGE_INDEX	New CLUT image index support
gcFEATURE_BIT_VG_PARALLEL_PATHS	New parallel path HW support
gcFEATURE_BIT_VG_PE_CLEAR	Pixel engine clear support
gcFEATURE_BIT_VG_PIXEL_MATRIX	Pixel matrix support
gcFEATURE_BIT_VG_QUALITY_8X	8x anti-aliasing path support
gcFEATURE_BIT_VG_RADIAL_GRADIENT	Radial gradient support
gcFEATURE_BIT_VG_RECTANGLE_TILED_OUT	Rectangle tiled output support
gcFEATURE_BIT_VG_RGBA2_FORMAT	RGBA2222 format support
gcFEATURE_BIT_VG_RGBA8_ETC2_EAC	ETC2/EAC compressed image format support
gcFEATURE_BIT_VG_SCISSOR	Scissor support
gcFEATURE_BIT_VG_SRC_PREMULTIPLIED	Source image alpha premultiplied
gcFEATURE_BIT_VG_STENCIL	Stencil image mode support
gcFEATURE_BIT_VG_STRIPE_MODE	Stripe mode support
gcFEATURE_BIT_VG_TESSELLATION_TILED_OUT	Tessellation tiled output support
gcFEATURE_BIT_VG_USE_DST	Read destination pixel support
gcFEATURE_BIT_VG_YUV_INPUT	YUV input format support
gcFEATURE_BIT_VG_YUV_OUTPUT	YUV format output support
gcFEATURE_BIT_VG_YUV_TILED_INPUT	YUV tiled input format support
gcFEATURE_BIT_VG_YUY2_INPUT	YUY2 input format support

Parent topic: Enumerations for product and feature queries

Parent topic:*Hardware product and feature information*

Structures for product and feature queries

This section describes structures used for product and feature queries.

vg_lite_info_t structure

This structure is used to query VGLite driver information.

Used in function: vg_lite_get_info_t.

vg_lite_info_t member	Type	Description
api_version	vg_lite_uint32_t	VGLite API version
header_version	vg_lite_uint32_t	VGLite header version
release_version	vg_lite_uint32_t	VGLite driver release version
reserved	vg_lite_uint32_t	Reserved for future use

Parent topic: Structures for product and feature queries

Parent topic:*Hardware product and feature information*

Functions for product and feature queries

This section describes functions used for product and feature queries.

vg_lite_get_product_info

Description:

This function is used to identify the VGLite-compatible product.

Syntax:

```
uint32_t vg_lite_get_product_info (
      char            *name,
      uint32_t        *chip_id,
      uint32_t        *chip_rev
);
```

Parameters:

Name	Description
name	A character array to store the name of the chip.
chip_id	Stores an ID number for the chip.
chip_rev	Stores a revision number for the chip.

Parent topic: Functions for product and feature queries

vg_lite_get_info

Description:

This function is used to query the VGLite driver information.

Syntax:

```
vg_lite_error_t vg_lite_get_info (
   vg_lite_info_t              *info
);
```

Parameters:

Name	Description
info	Points to the VGLite driver information structure, which includes the API version, header version, and release version

Parent topic: Functions for product and feature queries

vg_lite_get_register

Description:

This function can be used to read a GPU AHB register value given the AHB byte address of a register. Refer to the appropriate Vivante GPU AHB register specification documents for register descriptions. The value range of AHB accessible addresses for VGLite cores is usually 0x0 to 0x1FF and 0xA00 to 0xA7F.

Syntax:

```
vg_lite_error_t vg_lite_get_register (
      vg_lite_uint32_t      address,
      vg_lite_uint32_t      *result
);
```

Parameters:

Name	Description
address	Byte Address of the register which value you want.
*result	The registers value.

Parent topic: Functions for product and feature queries

vg_lite_query_feature

Description:

This function is used to query if a specific feature is available.

Syntax:

```
vg_lite_uint32_t vg_lite_query_feature (
   vg_lite_feature_t          feature
);
```

Parameters:

Name	Description
feature	Feature to be queried, as detailed in enum vg_lite_feature_t

Returns:

The feature is either not supported (0) or supported (1).

Parent topic: Functions for product and feature queries

vg_lite_get_mem_size

Description:

This function queries whether there is any remaining allocated contiguous video memory. *(available from June 2020)*

Syntax:

```
vg_lite_error_t vg_lite_get_mem_size(
   vg_lite_uint32_t             *size
);
```

Parameters:

Name	Description
size	Pointer to the remaining allocated contiguous video memory.

Returns:

Returns VG_LITE_SUCCESS if the query is successful and memory is available. Returns VG_LITE_NO_CONTEXT if the driver is not initialized or there is no available memory.

Parent topic: Functions for product and feature queries

Parent topic:*Hardware product and feature information*

API control Before calling any VGLite API function, the application must initialize the VGLite implicit (global) context by calling *vg_lite_init()*, which will fill in a features table, reset the fast-clear buffer, reset the compositing target buffer and allocate the command and tessellation buffers.

The VGLite driver only supports one current context and one thread to issue commands to the Vivante Vector Graphics hardware. The VGLite driver does not support multiple concurrent contexts running simultaneously in multiple threads/processes, as the VGLite kernel driver does not support context switching. A VGLite application can only use a single context at any time to issue commands to the Vivante Vector Graphics hardware. If a VGLite application must switch contexts, vg_lite_close() must be called to close the current context in the current thread, then vg_lite_init() can be called to initialize a new context either in the current thread or from another thread/process.

Context initialization and control functions

This section provides an overview of the context initialization and control functions.

vg_lite_init

Description:

This function initializes the memory and data structures needed for VGLite draw/blit functions, by allocating memory for the command buffer and a tessellation buffer of the specified size.

GC555 has a newly designed hardware tessellation module that requires less memory for the tessellation buffer than GC355 and GNanoLite-V. Specifically, the GC555 required tessellation buffer size is “buffer_height * 128 byte”. vg_lite_init API can simply be called with the render buffer “width” and “height” as the input parameters for GC555. This results in the best path to tessellation performance.

GC355 and GCNanoLiteV hardware tessellation module requires a tessellation buffer with size “buffer_height * buffer_width * 8 byte”. If system memory is limited, the application can define a smaller tessellation window based on the amount of memory available. GPU hardware can process the entire render buffer path tessellation in multiple passes with the tessellation window sliding across the render buffer. The multi-pass path tessellation with the smaller tessellation window has a certain performance overhead.

The minimum tessellation window that can be used is 16x16. If tess_height or tess_width is less than 0 in vg_lite_init API, then no path tessellation buffer is created and path drawing APIs do not work, only blit APIs can be used after vg_lite_init.

If this would be the first context that accesses the hardware, the hardware is turned on and initialized. If a new context must be initialized, vg_lite_close must be called to close the current context. Otherwise, vg_lite_init will return an error.

Syntax:

```
vg_lite_error_t vg_lite_init (
    vg_lite_int32_t             tess_width,
    vg_lite_int32_t             tess_height

);
```

Parameters:

Name	Description
tess_v	Width of tessellation window. Maximum cannot be greater than render buffer width. If less than or equal to 0, then no tessellation buffer is created, in which case only blit APIs can be used afterward.
tess_h	Height of tessellation window. Maximum cannot be greater than render buffer height. If less than or equal to 0, then no tessellation buffer is created, in which case blit APIs can be used afterward.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_close

Description:

This function deallocates all the resources and free up all the memory that was initialized earlier by the vg_lite_init function. It will also turn OFF the hardware automatically if this was the only active context.

Syntax:

```
vg_lite_error_t vg_lite_close (
    void
);
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_flush

Description:

This function explicitly submits the command buffer to the GPU without waiting for it to complete. *(From Dec 2019, return type is vg_lite_error_t, previously was void.)*

Syntax:

```
vg_lite_error_t vg_lite_flush (
    void
);
```

Returns:

Returns VG_LITE_SUCCESS if the flush is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_finish

Description:

This function explicitly submits the command buffer to the GPU and waits for it to complete.

Syntax:

```
vg_lite_error_t vg_lite_finish (
    void
);
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_frame_delimiter

Description:

This function sets a flag for GPU to signal the completion of current frame. A vg_lite_finish is called by default within this API. The enum VG_LITE_FRAME_END_FLAG is the only value that can be set by flag parameter.

Syntax:

```
vg_lite_error_t vg_lite_frame_delimiter (
    vg_lite_frame_flag_t flag
);
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Context initialization and control functions

vg_lite_set_command_buffer_size

Description:

This function is optional. If used, call it before vg_lite_init if you want to change the command buffer size. *(available from March 2020)*

Syntax:

```
vg_lite_error_t vg_lite_set_command_buffer_size (
    vg_lite_uint32_t            size
);
```

Parameters:

Name	Description
size	Size of the VGLite Command buffer. Default is 64K.

Returns:

Returns VG_LITE_SUCCESS if the flush is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_set_command_buffer

Description:

This function sets a user-defined external memory buffer (physical, 64-byte aligned) as the VGLite command buffer. By default, the VGLite driver allocates a static command buffer internally. Thus, it is not necessary for an application to allocate and set the command buffer. This function is only used for devices where an application needs to allocate the command buffer dynamically. *(from December 2021)*

Syntax:

```
vg_lite_error_t vg_lite_set_command_buffer (
    vg_lite_uint32_t            physical,
    vg_lite_uint32_t            size
);
```

Parameters:

Name	Description
physical	The physical address of a memory buffer. The address must be 64-byte aligned.
size	The size of memory buffer. The size must be 128-byte aligned.

Returns:

Returns VG_LITE_SUCCESS if the command buffer set is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_set_tess_buffer

Description:

This function specifies a memory buffer from an application as the VGLite driver's tessellation buffer. By default, the VGLite driver allocates a static tessellation buffer internally. Thus, it is not necessary for an application to allocate and set the tessellation buffer. This function is only used for devices where the application needs to allocate the tessellation buffer dynamically. *(from December 2021)*

Syntax:

```
vg_lite_error_t vg_lite_set_tess_buffer (
    vg_lite_uint32_t            physical,
    vg_lite_uint32_t            size
);
```

Parameters:

Name	Description
physical	The physical address of a tessellation buffer. The address must be 64-byte aligned.
size	The size of tessellation buffer. tessellation buffer size = target buffer's height * 128B.

Returns:

Returns VG_LITE_SUCCESS if the tessellation buffer set is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_set_memory_pool

Description:

This function sets the specific memory pool from which certain type of buffers, VG_LITE_COMMAND_BUFFER, VG_LITE_TESSELLATION_BUFFER, or VG_LITE_RENDER_BUFFER, should be allocated. By default, all types of buffers are allocated from VG_LITE_MEMORY_POOL_1. This API must be called before vg_lite_init() for setting VG_LITE_COMMAND_BUFFER or VG_LITE_TESSELLATION_BUFFER memory pools. This API can be called anytime for VG_LITE_RENDER_BUFFER to affect the following vg_lite_allocate() calls.*(from December 2023)*

Syntax:

```
vg_lite_error_t vg_lite_set_memory_pool (
    vg_lite_buffer_type_t        type,
    vg_lite_memory_pool_t        pool
);
```

Parameters:

Nam	Description
type	The buffer type (VG_LITE_COMMAND_BUFFER, VG_LITE_TESSELLATION_BUFFER, or VG_LITE_RENDER_BUFFER) to be allocated from memory pool.
pool	The memory pool (VG_LITE_MEMORY_POOL_1, VG_LITE_MEMORY_POOL_2) from which the buffer type should be allocated.

Returns:

Returns VG_LITE_SUCCESS if the memory pool set is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

Parent topic:*API control*

Pixel buffers This chapter provides an overview of the pixel buffer alignment, cache, internal representation, enumerations, structures, and functions.

Pixel buffer alignment

The VGLite hardware requires the pixel buffer start address and stride to be properly byte-aligned to work correctly. The start address and stride alignment requirement for a pixel buffer depends on the specific pixel format, and gcFEATURE_VG_16PIXELS_ALIGNED value (0/1) in vg_lite_options.h file.

Parent topic:*Pixel buffers*

Pixel cache

The Vivante Imaging Engine (IM) includes two fully associative caches. Each cache has 8 lines. Each line has 64 bytes. In this case, one cache line can hold either a 4x4-pixel tile or a 16x1-pixel row.

Parent topic:*Pixel buffers*

Internal representation

For non-32-bit color formats, each pixel is extended to 32 bits as follows:

If the source and destination formats have the same color format, but differ in the number of bits per color channel, the source channel is multiplied by (2d- 1)/(2s– 1) and is rounded to the nearest integer, where:

- **d** is the number of bits in the destination channel
- **s** is the number of bits in the source channel

Example: a b11111 5-bit source channel gets converted to an 8-bit destination b11111000.

The YUV formats are internally converted to RGB. The pixel selection is unified for all formats by using the LSB of the coordinate.

Parent topic:*Pixel buffers*

Pixel buffer enumerations

This section provides an overview of the pixel buffer enumerations.

vg_lite_buffer_format_t enumeration

This enumeration specifies the color format to use for a buffer. This applies to both image and Render Target. Formats include supported swizzles for RGB. For YUV swizzles, use the related values and parameters in vg_lite_swizzle_t.

The application shall use the vg_lite_query_feature API to determine support for some hardware-specific formats. For example, related vg_lite_feature_t enum values include gcFEATURE_BIT_VG_RGBA2_FORMAT and gcFEATURE_BIT_VG_IM_INDEX_FORMAT.

(Alignment columns refined March and Sept 2023)

Used in structure: vg_lite_buffer_t.

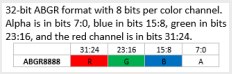
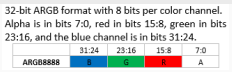
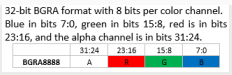
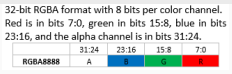
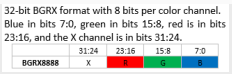
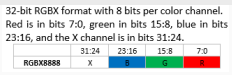
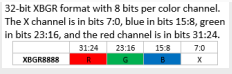
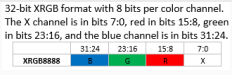
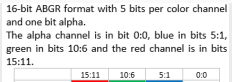
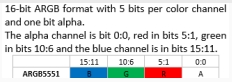
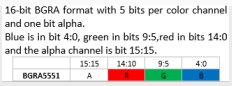
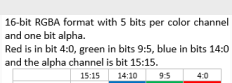
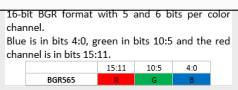
See also vg_lite_blit, vg_lite_clear, vg_lite_draw.

Attention: OpenVG VGImageFormat Note: The bits for each color channel are stored within a machine word from MSB to LSB in the order indicated by the pixel format name. This is the opposite of the original VG_LITE_* formats that are ordered from LSB to MSB. The formats with the same organization are listed in the same row as their VG_Lite counterparts.

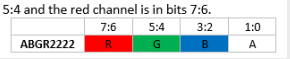
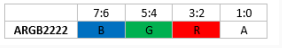
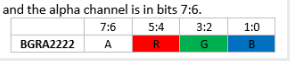
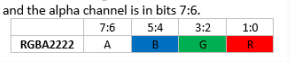
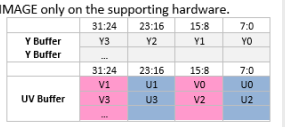
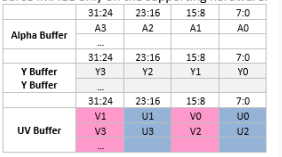
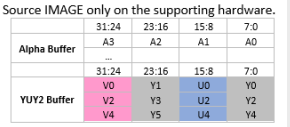
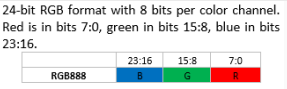
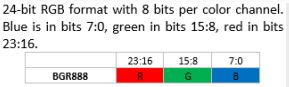
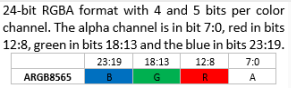
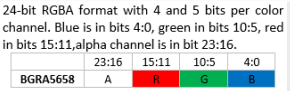
Attention: Original VGLite API Image Format Note: The bits for each color channel are stored within a machine word from LSB to MSB in the order indicated by the pixel format name. This is the opposite of the OPENVG VG_* formats that are ordered from MSB to LSB.

The following codes, as also used in OpenVG 1.1 Specification Table 11, are used for format description:

- A - Alpha channel
- B - Blue color channel
- G - Green color channel
- R - Red color channel
- X - Uninterpreted padding byte or bit
- L - Grayscale
- BW - 1-bit black and white
- l - Linear color space
- s - Non-linear (sRGB) color space
- PRE - Alpha values are premultiplied

vg_lite_buffer_format_t String Value	Description	Supported as source	Supported as destination	Start address/ Stride alignment: bytes
VG_LITE_ABGR8888 VG_sRGBA_8888 VG_sRGBA_8888_PRE VG_lRGBA_8888 VG_lRGBA_8888_PRE		Yes	Yes	Start 4B / Stride 64B
VG_LITE_ARGB8888 VG_sBGRA_8888 VG_sBGRA_8888_PRE VG_lBGRA_8888 VG_lBGRA_8888_PRE		Yes	Yes	Start 4B / Stride 64B
VG_LITE_BGRA8888 VG_sARGB_8888 VG_sARGB_8888_PRE VG_lARGB_8888 VG_lARGB_8888_PRE		Yes	Yes	Start 4B / Stride 64B
VG_LITE_RGBA8888 VG_sABGR_8888 VG_sABGR_8888_PRE VG_lABGR_8888 VG_lABGR_8888_PRE		Yes	Yes	Start 4B / Stride 64B
VG_LITE_BGRX8888 VG_sXRGB_8888 VG_lXRGB_8888		Yes	Yes	Start 4B / Stride 64B
VG_LITE_RGBX8888 VG_sXBGR_8888 VG_lXBGR_8888		Yes	Yes	Start 4B / Stride 64B
VG_LITE_XBGR8888 RGBX VG_sRGBX_8888 VG_lRGBX_8888		Yes	Yes	Start 4B / Stride 64B
VG_LITE_XRGB8888 VG_sBGRX_8888 VG_lBGRX_8888		Yes	Yes	Start 4B / Stride 64B
VG_LITE_ABGR1555 VG_sRGBA_5551		Yes	Yes	Start 4B / Stride 32B
VG_LITE_ARGB1555 VG_sBGRA_5551		Yes	Yes	Start 4B / Stride 32B
VG_LITE_BGRA5551 VG_sARGB_1555		Yes	Yes	Start 4B / Stride 32B
VG_LITE_RGBA5551 VG_sABGR_1555		Yes	Yes	Start 4B / Stride 32B
VG_LITE_BGR565 VG_sRGB_565		Yes	Yes	Start 4B / Stride 32B

16-bit RGB format with 5 or 6 bits per color channel.

Hardware-dependent formats for vg_lite_buffer_format_t	Description	Supported as source	Supported as destination	Alignment (bytes)
VG_LITE_ABGR2222	8-bit BGRA format with 2 bits per color channel. Alpha is in bits 1:0, blue in bits 3:2, green in bits 5:4 and the red channel is in bits 7:6. 	Yes	Yes	Start 4B / Stride 16B
VG_LITE_ARGB2222	8-bit BGRA format with 2 bits per color channel. Alpha is in bits 1:0, red in bits 3:2,green in bits 5:4 and the blue channel is in bits 7:6. 	Yes	Yes	Start 4B / Stride 16B
VG_LITE_BGRA2222	8-bit BGRA format with 2 bits per color channel. Blue is in bits 1:0, green in bits 3:2, red in bits 5:4 and the alpha channel is in bits 7:6. 	Yes	Yes	Start 4B / Stride 16B
VG_LITE_RGBA2222	8-bit RGBA format with 2 bits per color channel. Red is in bits 1:0, green in bits 3:2, blue in bits 5:4 and the alpha channel is in bits 7:6. 	Yes	No	8B
VG_LITE_INDEX_1	1-bit index format	Yes	No	8B
VG_LITE_INDEX_2	2-bit index format	Yes	No	both 8B
VG_LITE_INDEX_4	4-bit index format	Yes	No	both 8B
VG_LITE_INDEX_8	8-bit index format	Yes	No	both 16B
VG_LITE_NV12_TILED	Supertiled (8x8 pixels), planar YUV format, 96-bit for 4 pixels. Y plane is 32 bits for 4 pixels and is organized in 64 pixel super tiles (8x8 Y); UV plane is 64 bits for 4 pixels. Pixels are organized in super tiles are (4x4 UV pairs), available for Source IMAGE only on the supporting hardware. 	Yes	No	Y: both 16 Bytes UV: both 8 Bytes
VG_LITE_ANV12_TILED	Pixel organization as NV12_TILED but with an Alpha Buffer is also supertiled, available for Source IMAGE only on the supporting hardware. 	Yes	No	A, Y: both 16 Bytes UV: both 8 Bytes
VG_LITE_AYUY2_TILED	Supertiled (8x8) and packed YUV format a with separate tiled Alpha Buffer. Y0 is in bits 7:0 and V is in bits 31:23, available for Source IMAGE only on the supporting hardware. 	Yes	No	both 32B
VG_LITE_RGB888	24-bit RGB format with 8 bits per color channel. Red is in bits 7:0, green in bits 15:8, blue in bits 23:16. 	Yes	Yes	Start 4B / Stride 32B
VG_LITE_BGR888	24-bit RGB format with 8 bits per color channel. Blue is in bits 7:0, green in bits 15:8, red in bits 23:16. 		Yes	Yes
VG_LITE_ARGB8565	24-bit RGBA format with 4 and 5 bits per color channel. The alpha channel is in bit 7:0, red in bits 12:8, green in bits 18:13 and the blue in bits 23:19. 		Yes	Yes
VG_LITE_BGRA5658	24-bit RGBA format with 4 and 5 bits per color channel. Blue is in bits 4:0, green in bits 10:5, red in bits 15:11,alpha channel is in bit 23:16. 	Yes	Yes	Start 4B / Stride 32B

Parent topic: Pixel buffer enumerations

Image buffer alignment requirement

The image (or source) buffer alignment requirement depends on the specific pixel format, and some gcFEATURE_*_ALIGNED defines in the vg_lite_options.h file.

Image format	Bits per pixel	Source tile mode	Start address alignment requirement in bytes	Stric
VG_LITE_INDEX1	1	linear	8B	2B
VG_LITE_INDEX1	1	tile	8B	1B
VG_LITE_INDEX2	2	linear	8B	4B
VG_LITE_INDEX2	2	tile	8B	1B
VG_LITE_INDEX4	4	linear	8B	8B
VG_LITE_INDEX4	4	tile	8B	2B
VG_LITE_INDEX8	8	linear	16B	16B
VG_LITE_INDEX8	8	tile	16B	4B
VG_LITE_A4	4	linear	8B	8B
VG_LITE_A4	4	tile	8B	2B
VG_LITE_A8	8	linear	16B	16B
VG_LITE_A8	8	tile	16B	4B
VG_LITE_L8	8	linear	16B	16B
VG_LITE_L8	8	tile	16B	4B
VG_LITE_ARGB2222	8	linear	16B	16B
VG_LITE_ARGB2222	8	tile	16B	4B
VG_LITE_RGB565	16	linear	32B	32B
VG_LITE_RGB565	16	tile	32B	8B
VG_LITE_ARGB1555	16	linear	32B	32B
VG_LITE_ARGB1555	16	tile	32B	8B
VG_LITE_ARGB4444	16	linear	32B	32B
VG_LITE_ARGB4444	16	tile	32B	8B
VG_LITE_ARGB8888	32	linear	64B	64B
VG_LITE_ARGB8888	32	tile	64B	16B
VG_LITE_XRGB8888	32	linear	64B	64B
VG_LITE_XRGB8888	32	tile	64B	16B
VG_LITE_ARGB8565	24	linear	64B	48B*
VG_LITE_ARGB8565	24	tile	64B	12B*
VG_LITE_RGB888	24	linear	64B	48B*
VG_LITE_RGB888	24	tile	64B	12B*
VG_LITE_YUY2/UYVY	16	linear	32B	32B
VG_LITE_YUY2/UYVY	16	tile	32B	8B
VG_LITE_NV12	12	linear	Y: 32B UV: 32B	Y: 32
VG_LITE_YV12	12	linear	Y: 32B U: 16B V: 16B	Y: 32
VG_LITE_NV16	16	linear	Y: 32B UV: 32B	Y: 32
VG_LITE_YV16	16	linear	Y: 32B U: 16B V: 16B	Y: 32
VG_LITE_YV24	24	linear	Y: 32B U: 32B V: 32B	Y: 32
VG_LITE_ETC2	8	tile	16B	4B

Note:

1. The values in the table reflect the alignment requirements of the data in memory. The stride of ARGB8888 / ARGB8565 is seen as 4Byte per pixel when configuring the hardware.
2. For tile mode, the stride is still the byte size of a row of pixels in the buffer instead of 4 rows.
3. When DECNano function is enabled for the buffer, the total buffer size need align to 64Byte*compression rate for ARGB8888 or XRGB8888 format, align to 48Byte*compress rate for RGB888 format.

Additional Alignment Requirement

1. Buffer starting address must be 16 pixel-byte-size aligned, that is 8 bit-per-pixel format buffer must be 16 bytes aligned; 16 bit-per-pixel format buffer must be 32 bytes aligned; 24 and 32 bit-per-pixel format buffer must be 64 bytes aligned.
2. For linear mode buffer, the buffer stride must be 16 pixel-byte-size aligned.
3. For tile mode buffer, buffer width and height must be 4 pixel aligned so buffer width and height end at tile boundary.

Parent topic: Pixel buffer enumerations

Destination buffer alignment requirement

The destination (or render target) buffer alignment requirement depends on the specific pixel format, and some gcFEATURE_*_ALIGNED defines in the vg_lite_options.h file.

Target format	Bits per pixel	Target tile mode	VG tile mode	Start address alignment requirement i
VG_LITE_A8	8	linear	linear	4B
VG_LITE_A8	8	linear	tile	64B
VG_LITE_A8	8	tile	linear	64B
VG_LITE_A8	8	tile	tile	64B
VG_LITE_L8	8	linear	linear	4B
VG_LITE_L8	8	linear	tile	64B
VG_LITE_L8	8	tile	linear	64B
VG_LITE_L8	8	tile	tile	64B
VG_LITE_ARGB2222	8	linear	linear	4B
VG_LITE_ARGB2222	8	linear	tile	64B
VG_LITE_ARGB2222	8	tile	linear	64B
VG_LITE_ARGB2222	8	tile	tile	64B
VG_LITE_RGB565	16	linear	linear	4B
VG_LITE_RGB565	16	linear	tile	64B
VG_LITE_RGB565	16	tile	linear	64B
VG_LITE_RGB565	16	tile	tile	64B
VG_LITE_ARGB1555	16	linear	linear	4B
VG_LITE_ARGB1555	16	linear	tile	64B
VG_LITE_ARGB1555	16	tile	linear	64B
VG_LITE_ARGB1555	16	tile	tile	64B
VG_LITE_ARGB4444	16	linear	linear	4B
VG_LITE_ARGB4444	16	linear	tile	64B
VG_LITE_ARGB4444	16	tile	linear	64B
VG_LITE_ARGB4444	16	tile	tile	64B
VG_LITE_ARGB8888	32	linear	linear	4B
VG_LITE_ARGB8888	32	linear	tile	64B
VG_LITE_ARGB8888	32	tile	linear	64B
VG_LITE_ARGB8888	32	tile	tile	64B
VG_LITE_XRGB8888	32	linear	linear	4B
VG_LITE_XRGB8888	32	linear	tile	64B
VG_LITE_XRGB8888	32	tile	linear	64B
VG_LITE_XRGB8888	32	tile	tile	64B
VG_LITE_ARGB8565	24	linear	linear	64B
VG_LITE_ARGB8565	24	linear	tile	64B
VG_LITE_ARGB8565	24	tile	linear	64B
VG_LITE_ARGB8565	24	tile	tile	64B
VG_LITE_RGB888	24	linear	linear	64B
VG_LITE_RGB888	24	linear	tile	64B
VG_LITE_RGB888	24	tile	linear	64B
VG_LITE_RGB888	24	tile	tile	64B

Note:

1. The values in the table reflect the alignment requirements of pixel data in memory. The stride of ARGB8888/ARGB8565 is seen as 4 Bytes per pixel when configuring the hardware.
2. For tile mode, the buffer stride is still the byte size of a row of pixels instead of 4 rows of pixels.
3. For PE clear function, the clear size must align to 48 Bytes for the RGB888 or ARGB8565 format.
4. For PE clear function with DECNano enabled, the clear size must align to 48 Bytes for RGB888, align to 64 Bytes for ARGB8888 or XRGB8888.
5. If the DECNano function is enabled for the buffer, the target buffer start address needs to align to 64 Bytes.
6. If the DECNano function is enabled for the buffer, the total buffer size needs to align to a 64-byte compression rate for ARGB8888 or XRGB8888 format and align to a 48 Byte*compression rate for RGB888 format.

Additional Alignment Requirement

1. Buffer starting address must be at least 4-byte aligned. Buffer stride must be at least one pixel size aligned.
2. Buffer starting address must be 64-byte aligned for 24 bit-per-pixel format, or tile mode, or DECNano enabled.
3. Buffer height must be 4-pixel aligned for tile mode buffer.
4. For tile mode buffer, the buffer stride must be 16-byte aligned for non-24bit-per-pixel formats. So, 8 bits-per-pixel format buffer width must be 16-pixel aligned; 16 bits-per-pixel format buffer width must be 8-pixel aligned; 32 bit-per-pixel format buffer width must be 4 pixel aligned.
5. For tile mode buffer, the buffer stride must be 12-byte aligned for 24 bits-per-pixel formats, that is, the buffer width must be 4-pixel aligned.
6. For PE clear function, the clear size must align to 48 Bytes for 24-bits-per-pixel formats.
7. For PE clear function with DECNano enabled, the clear size must align to 48 Bytes for 24 bits-per-pixel formats and align to 64 Bytes for 32 bits-per-pixel formats.
8. If source buffer tile mode is different from destination buffer tile mode, buffer starting address must be 64 Byte aligned, buffer stride must be 64 Byte aligned for non-24 bits-per-pixel formats, buffer stride must be 48-Byte aligned for 24 bits-per-pixel formats.

VGLite hardware requires the raster image width to be a multiple of 16 pixels for linear gradient and radial gradient operations. This requirement applies to all image formats. Therefore, the user must pad an arbitrary image width to a multiple of 16 pixels for VGLite linear gradient and radial gradient APIs.

Parent topic: Pixel buffer enumerations

vg_lite_buffer_layout_t **enumeration**

Specifies the buffer data layout in memory.

Used in structure: vg_lite_buffer.

vg_lite_buffer_layout_t String Value	Description
VG_LITE_LINEAR	Linear (scanline) layout.
VG_LITE_TILED	Data is organized in 4x4 pixel tiles. **Note:** for this layout, the buffer start address and stride must be 64-byte aligned

Parent topic: Pixel buffer enumerations

vg_lite_compress_mode_t **enumeration**

Specifies the DECNano comprssion mode. *(from March 2023)*

Used in structure: vg_lite_buffer_t.

vg_lite_compress_mode_t string value	Description
VG_LITE_DEC_DISABLE	Disable compression.
VG_LITE_DEC_NON_SAMPLE	compression ratio is 1.6 for ARGB8888, 2.0 for XRGB8888
VG_LITE_DEC_HSAMPLE	compression ratio is 2.0 for ARGB8888, 2.6 for XRGB8888
VG_LITE_DEC_HV_SAMPLE	compression ratio is 2.6 for ARGB8888, 4.0 for XRGB8888

Parent topic: Pixel buffer enumerations

vg_lite_gamma_conversion_t **enumeration**

Specifies the gamma conversion mode *(from Sept 2022)*

Used in function: vg_lite_set_gamma.

vg_lite_gamma_conversion_t string value	Description
VG_LITE_GAMMA_NO_CONVERSION	Leave the color as it is.
VG_LITE_GAMMA_LINEAR	Convert from sRGB to linear.
VG_LITE_GAMMA_NON_LINEAR	Convert from linear to sRGB space.

Parent topic: Pixel buffer enumerations

vg_lite_index_endian_t **enumeration**

Specifies the endian order parsing mode for index formats *(from March 2023)*.

Used in structure: vg_lite_buffer_t.

vg_lite_index_endi string value	Description
VG_LITE_INDEX	Parse the index pixel from low to high, when using index1, the parsing order is bit0~bit7. when using index2, the parsing order is bit0:1,bit2:3,bit4:5.bit6:7. when using index4, the parsing order is bit0:3,bit4:7.
VG_LITE_INDEX	Parse the index pixel from low to high, when using index1, the parsing order is bit7~bit0. when using index2, the parsing order is bit7:6,bit5:4,bit3:2.bit1:0. when using index4, the parsing order is bit4:7,bit0:3.

Parent topic: Pixel buffer enumerations

vg_lite_image_mode_t enumeration

Specifies how an image is rendered onto a buffer *(prior to Sept 2022 name was* vg_lite_buffer_image_mode_t*)*.

Used in structure: vg_lite_buffer_t.

vg_lite_image_mode_t string value	Description
VG_LITE_ZERO	
VG_LITE_NORMAL_IMAGE_MODE	Image drawn with blending mode
VG_LITE_MULTIPLY_IMAGE_MODE	Image is multiplied with paint color
VG_LITE_STENCIL_MODE	
VG_LITE_NONE_IMAGE_MODE	Image input is ignored.
VG_LITE_RECOLOR_MODE	

Parent topic: Pixel buffer enumerations

vg_lite_map_flag_t enumeration

Specifies whether mapping is for user memory or the DMA buffer *(from March 2023)*.

Used in function: vg_lite_map.

vg_lite_map_flag_t string value	Description
VG_LITE_MAP_USER_MEMORY	Mapping is for user memory.
VG_LITE_MAP_DMABUF	Mapping is for the DMA buffer.

Parent topic: Pixel buffer enumerations

vg_lite_paint_type_t enumeration

Specifies paint type *(from May 2023)*.

Used in structure: vg_lite_buffer_t.

vg_lite_paint_type_t string value	Description
VG_LITE_PAINT_ZERO	
VG_LITE_PAINT_COLOR	Color
VG_LITE_PAINT_LINEAR_GRADIENT	Linear Gradient
VG_LITE_PAINT_RADIAL_GRADIENT	Radial Gradient
VG_LITE_PAINT_PATTERN	Pattern

Parent topic: Pixel buffer enumerations

vg_lite_transparency_t enumeration

Specifies the transparency mode for a buffer *(prior to Sept 2022 name was vg_lite_buffer_transparency_mode_t)*.

Used in structure:vg_lite_buffer.

vg_lite_transp string value	Description
VG_LITE_IM	Opaque image: all image pixels are copied to the VG PE for rasterization
VG_LITE_IM	Transparent image: only the non-transparent image pixels are copied to the VG PE. **Note:** This mode is only valid when IMAGE_MODE (vg_lite_image_mode_t) is either VG_LITE_NORMAL_IMAGE_MODE or VG_LITE_MULTIPLY_IMAGE_MODE.

Parent topic: Pixel buffer enumerations

vg_lite_swizzle_t enumeration

This enumeration specifies the swizzle for the UV components of YUV data.

Used in structure: vg_lite_yuvinfo.

vg_lite_swizzle_t string value	Description
VG_LITE_SWIZZLE_UV	U in lower bits, V in upper bits
VG_LITE_SWIZZLE_VU	V in lower bits, U in upper bits

Parent topic: Pixel buffer enumerations

vg_lite_yuv2rgb_t enumeration

This enumeration specifies the standard for conversion of YUV data to RGB data.

Used in structure: vg_lite_yuvinfo.

vg_lite_yuv2rgb_t string value	Description
VG_LITE_YUV601	YUV Converting with ITC.BT-601 standard
VG_LITE_YUV709	YUV Converting with ITC.BT-709 standard

Parent topic: Pixel buffer enumerations

Parent topic:*Pixel buffers*

Pixel buffer structures

This section provides an overview on the pixel buffer structures.

vg_lite_buffer_t structure

This structure defines the buffer layout for a VGLite image or memory data.

Used in structures: vg_lite_linear_gradient_t, vg_lite_radial_gradient_t.

Used in init functions: vg_lite_allocate, vg_lite_free, vg_lite_upload_buffer, vg_lite_map, vg_lite_unmap.

Used in blit functions:vg_lite_blit, vg_lite_blit_rect, vg_lite_clear, vg_lite_create_masklayer, vg_lite_fill_masklayer, vg_lite_blend_masklayer, vg_lite_set_masklayer, vg_lite_render_masklayer, vg_lite_destroy_masklayer

Used in draw functions: vg_lite_draw, vg_lite_draw_pattern, vg_lite_draw_grad, vg_lite_draw_radial_grad

vg_lite_buffer_t member	Type	Description
width	vg_lite_int32_t	Width of buffer in pixels
height	vg_lite_int32_t	Height of buffer in pixels
stride	vg_lite_int32_t	Stride in bytes
tiled	vg_lite_buffer_layo	Linear or tiled format for buffer enum
format	vg_lite_buffer_form	color format enum
handle	vg_lite_pointer	memory handle
memory	vg_lite_pointer	pointer to the start address of the memory
address	vg_lite_uint32_t	GPU address
yuv	vg_lite_yuvinfo_t	YUV format info struct
image_mode	vg_lite_image_mod	Blit image mode enum
transparency_mode	vg_lite_transparenc	Image transparency mode enum
fc_buffer[3]	vg_lite_fc_buffer_t	Three (3) fast clear buffers, reserved YUV format *(from March 2023)*
compress_mode	vg_lite_compress_n	Compression mode *(from March 2023)*
index_endian	vg_lite_index_endia	Big/Little Endian setting for index formats *(from March 2023)*
paintType	vg_lite_paint_type_	Paint type enum *(from May 2023)*
fc_enable	vg_lite_int8_t	Enable Image fast clear *(moved from Aug 2023)*
scissor_layer	vg_lite_int8_t	Get paintcolor from different paint types *(from Aug 2023)*
premulitplied	vg_lite_int8_t	The RGB pixel values are alpha-premultipled *(from Aug 2023)*

Parent topic: Pixel buffer structures

vg_lite_fc_buffer_t **structure**

This structure defines the organization of a fast clear buffer. *(from March 2023)*

Used in structure: vg_lite_buffer_t.

vg_lite_fc_buffer_t members	Type	Description
width	vg_lite_int32_t	Width of buffer in pixels
height	vg_lite_int32_t	Height of buffer in pixels
stride	vg_lite_int32_t	Stride in bytes
handle	vg_lite_pointer	memory handle as allocated by the VGLite kernel
memory	vg_lite_pointer	logical pointer to the start address of the memory for the CPU
address	vg_lite_uint32_	address to the buffer's memory for the GPU hardware
color	vg_lite_uint32_	The fast clear color value

Parent topic: Pixel buffer structures

vg_lite_yuvinfo_t **structure**

This structure defines the organization of VGLite YUV data.

Used in structure: vg_lite_buffer_t.

vg_lite_yuvinfo_t member	Type	Description
swizzle	vg_lite_swizzle_t	UV swizzle enum
yuv2rgb	vg_lite_yuv2rgb_	YUV conversion standard enum
‘uv_planar	vg_lite_uint32_t	UV (U) planar address for GPU, generated by driver
v_planar	vg_lite_uint32_t	V planar address for GPU, generated by driver
‘alpha_planar	vg_lite_uint32_t	Alpha planar address for GPU, generated by driver
‘uv_stride	vg_lite_uint32_t	UV (U) stride in bytes
‘v_stride	vg_lite_uint32_t	V planar stride in bytes
alpha_stride	vg_lite_uint32_t	Alpha stride in bytes
‘uv_height	vg_lite_uint32_t	UV (U) height in pixels
‘v_height	vg_lite_uint32_t	V stride in bytes
uv_memory	vg_lite_pointer	Logical pointer to the UV (U) planar memory
‘v_memory	vg_lite_pointer	Logical pointer to the V planar memory
uv_handle	vg_lite_pointer	Memory handle of the UV (U) planar, generated by the driver
v_handle	vg_lite_pointer	Memory handle of the V planar, generated by the driver

Parent topic: Pixel buffer structures

Parent topic:*Pixel buffers*

Pixel buffer functions

This section provides an overview of the pixel buffer functions.

vg_lite_allocate function

Description:

This function is used to allocate a buffer before it is used in either blit or draw functions.

To allow the hardware to access some memory, such as a source image or target buffer, you must first allocate the memory. The supplied vg_lite_buffer_t structure must be initialized with the size (width and height) and format of the requested buffer. If the stride is set to zero, then this function fills it in. The only input parameter to this function is the pointer to the buffer structure. If the structure has all the information needed, then appropriate memory is allocated for the buffer.

This function calls the kernel to allocate the memory. The kernel fills in the memory handle, logical address, and hardware addresses in the vg_lite_buffer_t structure.

Alignment note:

Vivante GPUs have an alignment requirement of 64 bytes. However, to meet the alignment requirements of the Vivante display controller, the VGLite driver sets the render target buffer alignment to 128 bytes. For source image buffer alignment requirements, see the alignment notes available in Table 1.

The vg_lite_buffer_format_t value descriptions:

Syntax:

```
vg_lite_error_t vg_lite_allocate (
    vg_lite_buffer_t    *buffer
);
```

Parameters:

Nam	Description
buffe	Pointer to the buffer that holds the size and format of the buffer being allocated. Either the memory or address field must be set to a non-zero value to map either a logical or physical address into hardware accessible memory.

Returns:

- VG_LITE_SUCCESS if the contiguous buffer was allocated successfully.
- VG_LITE_OUT_OF_RESOURCES if there is insufficient memory in the host OS heap for the buffer.
- VG_LITE_OUT_OF_MEMORY if allocation of a contiguous buffer failed.

Parent topic: Pixel buffer functions

vg_lite_free **function**

Description:

This function is used to deallocate the buffer that was previously allocated. It frees up the memory for that buffer.

Syntax:

```
vg_lite_error_t vg_lite_free (
      vg_lite_buffer_t       *buffer
);
```

Parameters:

Name	Description
buffer	Pointer to a buffer structure that was filled in by calling the vg_lite_allocate() function.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_upload_buffer **function**

Description:

The function uploads the pixel data to a GPU memory buffer object. The format of the data (pixel) to be uploaded must match the format defined for the buffer object. The input data memory buffer should contain enough data to be uploaded to the GPU buffer pointed by the input parameter buffer.

Note: Vivante Vector Graphics IP only uses data[0] and stride[0] as it does not support planar YUV formats..

Syntax:

```
vg_lite_error_t vg_lite_upload_buffer (
    vg_lite_buffer_t           *buffer,
    vg_lite_uint8_t            *data[3],
    vg_lite_uint32_t           stride[3]
);
```

Parameters:

Name	Description
buffer	Pointer to a buffer structure that was filled in by calling the vg_lite_allocate() function
data[3]	Pointer to pixel data. For the YUV format, there may be up to 3 pointers.
stride[3]	Stride for the pixel data

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_map function

Description:

This function is used to map the memory appropriately for a particular buffer. For some operating systems, it is used to get proper translation to the physical or logical address of the buffer needed by the GPU.

To use a frame buffer directly as a target buffer:

- Wrap a vg_lite_buffer_t structure around the buffer
- Call the kernel to map the supplied logical or physical address into hardware accessible memory

For example, if you know the logical address of the frame buffer, set the memory field of the vg_lite_buffer_t structure with that address and call this function. If you know the physical address, set the memory field to NULL and program the address field with the physical address.

Syntax:

```
vg_lite_error_t  vg_lite_map  (
    vg_lite_buffer_t          *buffer,
    vg_lite_map_flag_t        flag,
    int32_t                   fd
    );
```

Parameters:

Name	Description
*buffe	Pointer to a buffer structure that was filled in by calling the vg_lite_allocate() function
flag	Enumerate the vg_lite_map_flag_t value that specifies whether the mapping is for user memory or DMA buffer. *(from March 2023)*
fd	File descriptor for dma_buf if the flag is VG_LITE_MAP_DMABUF. Otherwise, this parameter is ignored. *(from March 2023)*

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_unmap function

Description:

This function unmaps the buffer and frees any memory resources allocated by a previous call to the vg_lite_map() function.

Syntax:

```
vg_lite_error_t vg_lite_unmap (
      vg_lite_buffer_t        *buffer
);
```

Parameters:

Name	Description
buffer	Pointer to a buffer structure that was filled in by calling the vg_lite_map() function

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_flush_mapped_buffer function

Description:

This function flushes the CPU cache for the mapped buffer to make sure the buffer contents are written to GPU memory.

Syntax:

```
vg_lite_error_t vg_lite_flush_mapped_buffer (
    vg_lite_buffer_t            *buffer
);
```

Parameters:

Name	Description
*buffer	Pointer to a buffer structure that was filled in by calling the vg_lite_map() function

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_set_CLUT function

Description:

This function sets the Color Lookup Table (CLUT) in the context state for index color image. Once the CLUT is set (Not NULL), the image pixel color for index format image rendering is obtained from the Color Lookup Table (CLUT) according to the pixel's color index value.

Note: Available only for IP with Indexed color support..

Syntax:

```
vg_lite_error_t vg_lite_set_CLUT (
  vg_lite_uint32_t                  count,
  vg_lite_uint32_t                  *colors
);
```

Parameters:

Nam	Description
coun	This is the count of the colors in the color look-up table: - For INDEX_1, there can be up to 2 colors in the table - For INDEX_2, there can be up to 4 colors in the table - For INDEX_4, there can be up to 16 colors in the table - For INDEX_8, there can be up to 256 colors in the table
*col-ors	The Color Lookup Table (CLUT) pointed by “colors” will be stored in the context and programmed to the command buffer when needed. The CLUT will not take effect until the command buffer is submitted to HW. The color is in ARGB format with A located in the upper bits. **Note:** The VGLite driver does not validate the CLUT contents from the application.

Returns:

VG_LITE_SUCCESS as no checking is done.

Parent topic: Pixel buffer functions

vg_lite_enable_dither **function**

Description:

This function is used to enable the dither function. Dither is turned off by default. The application can use the VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_DITHER) to determine HW support for dither.

Syntax:

```
vg_lite_error_t vg_lite_enable_dither (
);
```

Parameters: None

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_disable_dither **function**

Description:

This function is used to disable the dither function. Dither is turned off by default.

Syntax:

```
vg_lite_error_t vg_lite_disable_dither (
);
```

Parameters: None

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_set_gamma **function**

Description:

This function sets a gamma value.

Application can use the VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_GAMMA) to determine HW support for gamma.

Syntax:

```
vg_lite_error_t vg_lite_set_gamma (
    vg_lite_gamma_conversion_t     gamma_value
);
```

Parameters:

Name	Description
gamma_value	Sets a gamma value. See enum vg_lite_gamma_conversion_t.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

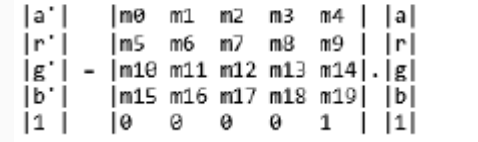
Parent topic:*Pixel buffers*

Matrices

This part of the API provides matrix controls.

Note: All the transformations in the driver/API are actually the final plane/surface coordinate system. There is no transformation of different coordinate systems with VGLite.

Matrix control float parameter type

Name	Typedef	Value
vg_lite_float_t	float	A single-precision floating-point number
vg_lite_pixel_matrix_t [20]	vg_lite_float_t	Pixel transform matrix m[20], which transforms each pixel as follows: 

Parent topic:*Matrices*

Matrix control structures

This section provides an overview of the graphic transformation matrix control structures.

vg_lite_matrix_t structure

This structure defines a 3x3 floating point matrix.

Used in structures: vg_lite_linear_gradient_t, vg_lite_radial_gradient_t.

Used in blit functions: vg_lite_blit, vg_lite_blit_rect.

Used in draw functions: vg_lite_draw, vg_lite_draw_gradient, vg_lite_draw_radial_gradient, vg_lite_draw_pattern, vg_lite_identity, vg_lite_scale, vg_lite_translate.

vg_lite_matrix_t member	Type	Description
m[3][3]	vg_lite_float_t	3x3 matrix, in [row] [column] order

Parent topic: Matrix control structures

vg_lite_pixel_channel_enable_t structure

This structure provides enable disable flags for hardware pixel channels A,R,G,B.

Used in function: vg_lite_set_pixel_matrix_t.

vg_lite_pixel_channel_enable_t members	Type	Description
enable_a	vg_lite_uint8_t	Enable A channel
enable_b	vg_lite_uint8_t	Enable B channel
enable_g	vg_lite_uint8_t	Enable G channel
enable_r	vg_lite_uint8_t	Enable R channel

Parent topic: Matrix control structures

Parent topic:*Matrices*

Matrix control functions

This section provides an overview of the matrix control functions.

vg_lite_identity function

Description:

This function loads an identity matrix into a matrix variable.

Syntax:

```
vg_lite_error_t vg_lite_identity (
    vg_lite_matrix_t            *matrix,
);
```

Parameters:

Name	Description
*matrix	Pointer to the vg_lite_matrix_t structure that will be loaded with an identity matrix.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Matrix control functions

vg_lite_set_pixel_matrix function

Description:

This function sets up a pixel transform matrix m[20] which transforms each pixel as follows:

```
|a'|    |m0  m1  m2  m3  m4 | |a|
|r'|    |m5  m6  m7  m8  m9 | |r|
|g'| =  |m10 m11 m12 m13 m14|.|g|
|b'|    |m15 m16 m17 m18 m19| |b|
|1 |    |0   0   0   0   1  | |1|
```

The pixel transform for the A, R, G, B channels can be enabled/disabled individually with the channel parameter.

Applications can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_PIXEL_MATRIX) to determine HW support for gaussian blur.

Syntax:

```
vg_lite_error_t vg_lite_set_pixel_matrix (
    vg_lite_pixel_matrix_t          matrix,
    vg_lite_pixel_channel_enable_t  *channel
);
```

Parameters:

Name	Description
*matrix	Specifies the vg_lite_pixel_matrix_t pixel transform matrix that will be loaded.
*channel	Pointer to the vg_lite_pixel_channel_enable_t structure used to enable/disable individual channels.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Matrix control functions

vg_lite_rotate function

Description:

This function rotates a matrix a specified number of degrees.

Syntax:

```
vg_lite_error_t vg_lite_rotate (
    vg_lite_float_t          degrees,
    vg_lite_matrix_t         *matrix
);
```

Parameters:

Nam	Description
degrees	Number of degrees to rotate the matrix. Positive numbers rotate clockwise.The coordinates for the transformation are given in the surface coordinate system (top-to-bottom orientation). Rotations with positive angles are in the clockwise direction.
*matrix	Pointer to the vg_lite_matrix_t structure that has to be rotated

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Matrix control functions

vg_lite_scale function

Description:

This function scales a matrix in both horizontal and vertical directions.

Syntax:

```
vg_lite_error_t vg_lite_scale (
    vg_lite_float_t             scale_x,
    vg_lite_float_t             scale_y,
    vg_lite_matrix_t            *matrix
);
```

Parameters:

Name	Description
scale_x	Horizontal scale
scale_y	Vertical scale
matrix	Pointer to the vg_lite_matrix_t structure that will be scaled.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Matrix control functions

vg_lite_translate function

Description:

This function translates a matrix to a new location.

Syntax:

```
vg_lite_error_t vg_lite_translate (
    vg_lite_float_t             x,
    vg_lite_float_t             y,
    vg_lite_matrix_t            *matrix
);
```

Parameters:

Name	Description
x	X location of the transformation.
y	Y location of the transformation.
matrix	Pointer to the vg_lite_matrix_t structure that will be translated.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Matrix control functions

Parent topic:*Matrices*

Blits for compositing and blending This part of the API performs the hardware accelerated blit operations.

Compositing rules describes how two areas are combined to form a single area. Blending rules describes how combining the colors of the overlapping areas are combined. VGLite supports two blending operations and a subset of the Porter-Duff operations [PD84]. The Porter-Duff operators assume that the pixels have the alpha associated (premultiplied), it means that the pixels are premultiplied prior to the blending operation. GC555, GC355, and some GCNanoUltraV hardware support alpha premultiply for RGB image, but GCNanoLiteV does not.

The source image is copied to the destination window with a specified matrix that can include translation, rotation, scaling, and perspective correction.

- The blit function can be used with or without the blend mode.
- The blit function can be used with or without specifying any color value.
- The blit function can be used for color conversion with an identity matrix and appropriate formats specified for the source and the destination buffers. In this case, do not specify blend mode and color value.

Blit enumerations

This section gives details on blit enumerations.

vg_lite_blend_t enumeration

This enumeration defines the blending modes supported by some VGLite API functions. S and D represent source and destination non-premultiplied RGB color channels. Sa and Da represent the source and destination alpha channels. SP and DP represent source and destination alpha-premultiplied RGB color channels (SP = S*Sa, DP = D*Da).

Note: VG_LITE_BLEND_*_LVGL modes are supported on all VG cores. On VG cores that do not support gcFEATURE_BIT_VG_LVGL_SUPPORT, the LVGL blend modes are supported by a combination of software and hardware operations. OPENVG_BLEND_* modes can only be supported on GC355 and GC555 cores.

Used in blit functions: vg_lite_blit, vg_lite_blit2, vg_lite_blit_rect.

Used in draw functions: vg_lite_draw, vg_lite_draw_grad, vg_lite_draw_radial_grad, vg_lite_draw_pattern.

vg_lite_blend_t String Values	Description
VG_LITE_BLEND_NONE	**S, no blending** Non-premultiplied
VG_LITE_BLEND_SRC_OVER	**S** + **D * (1 - Sa)** Non-premultiplied
VG_LITE_BLEND_DST_OVER	**S * (1 – Da)** + **D** Non-premultiplied
VG_LITE_BLEND_SRC_IN	**S * Da** Non-premultiplied
VG_LITE_BLEND_DST_IN	**D * Sa** Non-premultiplied
VG_LITE_BLEND_MULTIPLY	**S * (1 - Da)** + **D * (1 - Sa)** + **S * D** Non-premultiplied
VG_LITE_BLEND_SCREEN	**S** + **D - S * D** Non-premultiplied
VG_LITE_BLEND_DARKEN	**min(SRC_OVER, DST_OVER)** Non-premultiplied
VG_LITE_BLEND_LIGHTEN	**max(SRC_OVER, DST_OVER)** Non-premultiplied
VG_LITE_BLEND_ADDITIVE	**S** + **D** Non-premultiplied
VG_LITE_BLEND_SUBTRACT	**D * (1 - Sa)** Non-premultiplied
VG_LITE_BLEND_NORMAL_LVGL	**S * Sa** + **D * (1 - Sa)** Non-premultiplied (from March 2023)
VG_LITE_BLEND_ADDITIVE_LVGL	**(S** + **D) * Sa** + **D * (1 - Sa)** Non-premultiplied (from March 2023)
VG_LITE_BLEND_SUBTRACT_LVGL	**(S - D) * Sa** + **D * (1 - Sa)** Non-premultiplied (from March 2023)
VG_LITE_BLEND_MULTIPLY_LVGL	**(S * D) * Sa** + **D * (1 - Sa)** Non-premultiplied (from March 2023)
OpenVG Porter-Duff Blend String Values	*(from Aug 2023)*
OPENVG_BLEND_NONE	**SP, no blending** Premultiplied
OPENVG_BLEND_SRC_OVER	**(SP** + **DP * (1 - Sa))** / **(Sa** + **Da * (1 - Sa))** Premultiplied
OPENVG_BLEND_DST_OVER	**(SP * (1 - Da)** + **DP)** / **(Sa * (1 - Da)** + **Da)** Premultiplied
OPENVG_BLEND_SRC_IN	**(SP * Da)** / **(Sa * Da)** Premultiplied
OPENVG_BLEND_DST_IN	**(DP * Sa)** / **(Sa * Da)** Premultiplied
OPENVG_BLEND_MULTIPLY	**(SP*DP** + **SP*(1 - Da)** + **DP*(1 - Sa))** / **(Sa** + **Da*(1 - Sa))** Premultiplied
OPENVG_BLEND_SCREEN	**(SP** + **DP - (SP*DP))** / **(Sa** + **Da*(1 - Sa))** Premultiplied
OPENVG_BLEND_DARKEN	**min(SRC_OVER, DST_OVER)** Premultiplied
OPENVG_BLEND_LIGHTEN	**max(SRC_OVER, DST_OVER)** Premultiplied
OPENVG_BLEND_ADDITIVE	**(SP** + **DP)** / **(Sa** + **Da)** Premultiplied

Parent topic: Blit enumerations

vg_lite_color_t **parameter**

The common parameter vg_lite_color_t is described in Table 1.

Parent topic: Blit enumerations

vg_lite_color_transform_t **structure**

Specifies the pixel color_transform values for scale and bias.

Used in functions: vg_lite_set_color_transform.

vg_lite_color_transform_t members	Type	Description
a_scale	vg_lite_float_t	Scale value for alpha.
a_bias	vg_lite_float_t	Bias value for alpha.
r_scale	vg_lite_float_t	Scale value for red.
r_bias	vg_lite_float_t	Bias value for red.
g_scale	vg_lite_float_t	Scale value for green.
g_bias	vg_lite_float_t	Bias value for green.
b_scale	vg_lite_float_t	Scale value for blue.
b_bias	vg_lite_float_t	Bias value for blue.

Parent topic: Blit enumerations

vg_lite_filter_t **enumeration**

Specifies the sample-filtering mode in VGLite blit and draw APIs.

Used in blit functions: vg_lite_blit, vg_lite_blit_rect.

Used in draw functions: vg_lite_draw_radial_gradient, vg_lite_draw_pattern.

vg_lite_filter_t string values	Description
VG_LITE_FILTER_POINT	Fetch only the nearest image pixel
VG_LITE_FILTER_LINEAR	Use linear interpolation along a horizontal line
VG_LITE_FILTER_BI_LINEAR	Use a 2x2 box around the image pixel and perform an interpolation
VG_LITE_FILTER_GAUSSIAN	Perform 3x3 gaussian blur with the convolution for image pixel. *(from March 2023)*

Parent topic: Blit enumerations

vg_lite_color_transform_t **structure**

Specifies the pixel color_transform values for scale and bias.

Used in functions: vg_lite_set_color_transform.

vg_lite_color_transform_t members	Type	Description
a_scale	vg_lite_float_t	Scale value for alpha.
a_bias	vg_lite_float_t	Bias value for alpha.
r_scale	vg_lite_float_t	Scale value for red.
r_bias	vg_lite_float_t	Bias value for red.
g_scale	vg_lite_float_t	Scale value for green.
g_bias	vg_lite_float_t	Bias value for green.
b_scale	vg_lite_float_t	Scale value for blue.
b_bias	vg_lite_float_t	Bias value for blue.

Parent topic: Blit enumerations

vg_lite_mask_operation_t **enumeration**

Specifies the mask operation mode in VGLite blit APIs.

Used in functions: vg_lite_blend_masklayer, vg_lite_render_masklayer.

vg_lite_mask_o string values	Description
VG_LITE_CL	This operation sets all mask values in the region of interest to 0, ignoring the new mask layer.
VG_LITE_FIL	This operation sets all mask values in the region of interest to 1, ignoring the new mask layer.
VG_LITE_SE	This operation copies values in the region of interest from the new mask layer, overwriting the previous mask values.
VG_LITE_UN	This operation replaces the previous mask in the region of interest by its union with the new mask layer. The resulting values are always greater than or equal to their previous value.
VG_LITE_INT	This operation replaces the previous mask in the region of interest by its intersection with the new mask layer. The resulting mask values are always less than or equal to their previous value.
VG_LITE_SUI	This operation subtracts the new mask from the previous mask and replaces the previous mask in the region of interest by the resulting mask. The resulting values are always less than or equal to their previous value.

Parent topic: Blit enumerations

vg_lite_orientation_t **enumeration**

Specifies the mirror orientation in VGLite blit APIs.

Used in functions: vg_lite_set_mirror.

vg_lite_orientation_t string values	Description
VG_LITE_ORIENTATION_TOP_BOTT	Target output orientation is from top to bottom (default).
VG_LITE_ORIENTATION_BOTTOM_T	Target output orientation is from bottom to top.

Parent topic: Blit enumerations

vg_lite_param_type_t **enumeration**

Specifies the parameter type in VGLite blit APIs.

Used in functions: vg_lite_get_parameter.

vg_lite_param_type_t string value	Description
VG_LITE_GPU_IDLE_STATE	The count must be 1 for GPU idle state TRUE or FALSE.
VG_LITE_SCISSOR_RECT	The count must be 4n for x, y, right, bottom.

Parent topic: Blit enumerations

Parent topic:*Blits for compositing and blending*

Blit structures

This section provides details about blit structures.

vg_lite_buffer_t **structure**

Defined under the “Pixel buffer structures” section (see vg_lite_buffer_t structure).

Parent topic: Blit structures

vg_lite_color_key_t **structure**

A “color key” have two sections, where each section contains R,G,B channels, which are noted as high_rgb and low_rgb respectively. *(from April 2022)*

When the enable value is true, the color key specified is effective and the alpha value is used to replace the alpha channel of the destination pixel when its RGB channels are in range [low_rgb, high_rgb]. After the color key is used in the current frame, if the color key is not needed for the next frame, it should be disabled before the next frame.

Used in structure: vg_lite_color_key4_t

vg_lite_color_key_t members	Type	Description
enable	vg_lite_uint8	When set (true), this color key is enabled
low_r	vg_lite_uint8	The R channel of low_rgb
low_g	vg_lite_uint8	The G channel of low_rgb
low_b	vg_lite_uint8	The B channel of low_rgb
alpha	vg_lite_uint8	The alpha value to replace the destination pixel alpha channel value with
high_r	vg_lite_uint8	The R channel of high_rgb
high g	vg_lite_uint8	The G channel of high_rgb
high_b	vg_lite_uint8	The B channel of high_rgb

Parent topic: Blit structures

vg_lite_color_key4_t **structure**

The priority order is: color_key_0 > color_key_1 > color_key_2 > color_key_3. *(from April 2022)*

Used in blit function: vg_lite_set_color_key

vg_lite_color_key4_t members	Type	Description
color_key_0		high_rgb_0, low_rgb_0, alpha_0, enable_0
color_key_1		high_rgb_1, low_rgb_1, alpha_1, enable_1
color_key_2		high_rgb_2, low_rgb_2, alpha_2, enable_2
color_key_3		high_rgb_3, low_rgb_3, alpha_3, enable_3

Parent topic: Blit structures

vg_lite_matrix_t **structure**

Defined under the “Matrix control structures” section (see vg_lite_matrix_t structure).

Parent topic: Blit structures

vg_lite_path_t **structure**

Defined under the “Vector path structures” section (see vg_lite_path_t structure).

Parent topic: Blit structures

vg_lite_rectangle_t **structure**

This structure defines a rectangle by using coordinates.

Used in blit function: vg_lite_clear.

vg_lite_rectangle_t member	Type	Description
x	vg_lite_int32_t	X origin of rectangle, left coordinate in pixels
y	vg_lite_int32_t	Y origin of rectangle, top coordinate in pixels
width	vg_lite_int32_t	X Width of rectangle in pixels
height	vg_lite_int32_t	Y Height of rectangle in pixels

Parent topic: Blit structures

vg_lite_point_t **structure**

This structure defines a 2D point *(from March 2021)*.

Used in structure: vg_lite_point4_t.

vg_lite_point_t member	Type	Description
X	vg_lite_int32_t	X value of coordinate
Y	vg_lite_int32_t	Y value of coordinate

Parent topic: Blit structures

vg_lite_point4_t **structure**

This structure defines four 2D points that form a polygon. The points are defined by structure vg_lite_point_t. *(from March 2021)*

vg_lite_point4_t member	Type	Description
vg_lite_point_t[4]	vg_lite_int32_t each	a set of four points

Parent topic: Blit structures

vg_lite_float_point_t **structure**

This structure defines a 2D float point *(from March 2024)*.

Used in structure: vg_lite_float_point4_t.

vg_lite_float_point_t members	Type	Description
x	vg_lite_float_t	X value of coordinate
y	vg_lite_float_t	Y value of coordinate

Parent topic: Blit structures

vg_lite_float_point4_t **structure**

This structure defines four 2D float points that form a polygon. The points are defined by structure vg_lite_float_point_t. *(from March 2024)*

Used in blit function: vg_lite_get_transform_matrix.

vg_lite_float_point4_t members	Type	Description
vg_lite_float_point[4]	vg_lite_float_t each	a set of four points

Parent topic: Blit structures

Parent topic:*Blits for compositing and blending*

Blit functions

This section provides an overview on blit functions.

vg_lite_blit function

Description:

This is the blit function. The blit operation is performed using a source and a destination buffer. The source and destination buffer structures are defined using the vg_lite_buffer_t structure. Blit copies a source image to the destination window with a specified matrix that can include translation, rotation, scaling, and perspective correction. Note that vg_lite_buffer_t does not support coverage sample anti-aliasing so the destination buffer edge may not be smooth, especially with a rotation matrix. VGLite path rendering can be used to achieve high-quality coverage sample anti-aliasing (16X, 8X, 4X) rendering effect.

Note:

- The blit function can be used with or without the blend function (vg_lite_blend_t)
- The blit function can be used with or without specifying a foreground color value (vg_lite_color_t)
- The blit function can be used for color conversion with an identity matrix and appropriate formats specified for the source and the destination buffers. In this case, do not specify blend mode and color value.

Syntax:

```
vg_lite_error_t vg_lite_blit (
    vg_lite_buffer_t      *target,
    vg_lite_buffer_t      *source,
    vg_lite_matrix_t      *matrix,
    vg_lite_blend_t       blend,
    vg_lite_color_t       color,
    vg_lite_filter_t      filter
);
```

Parameters:

Nam	Description
*tar-get	Points to the vg_lite_buffer_t structure, which defines the destination buffer. See Image Source Alignment Requirement for valid destination color formats for the blit functions.
*sou	Points to the vg_lite_buffer_t structure for the source buffer. All color formats available in the vg_lite_buffer_format_t enum are valid source formats for the blit function.
*ma-trix	Points to a vg_lite_matrix_t structure that defines the transformation matrix of source pixels into the target. If the matrix is NULL, then an identity matrix is assumed, which means that the source is copied directly at 0,0 location on the target.
blen	Specifies one of the enum vg_lite_blend_t values for hardware-supported blend modes to be applied to each image pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0). **Note:** If the matrix parameter is specified with rotation or perspective, and the blend parameter is specified as VG_LITE_BLEND_NONE, VG_LITE_BLEND_SRC_IN, or VG_LITE_BLEND_DST_IN; then, the VGLite driver overwrites the application setting for the blit operation as follows: - If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is supported, then Transparency mode is always set to TRANSPARENT- If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is not supported, then Blend mode is always set to VG_LITE_BLEND_SRC_OVER. It happens due to some limitations in the VGLite hardware.
color	If non-zero, this color value is used as a mix color. The mixed color gets multiplied with each source pixel before blending happens. If you don't need a mix color, set the color parameter to 0.**Note:** this parameter has no effect if the source vg_lite_buffer_t structure member image_mode is set to VG_LITE_ZERO or VG_LITE_NORMAL_IMAGE_MODE.
fil-ter	Specifies the filter type. All formats available in the vg_lite_filter_t enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit functions

vg_lite_blit2 function

Description:

This is the blit function for use with two sources. The blit2 operation is performed using two source buffers and one destination buffer. The source and destination buffer structures are defined using the vg_lite_buffer_t structure. Source0 and Source1 are first blended according to the blend mode with a specific transformation matrix for each image. Source1 is used as the source while Source0 is used as the dest and is directly output to the render target buffer.

The specified matrices can include translation, rotation, scaling, and perspective correction. Note that vg_lite_buffer_t does not support coverage sample anti-aliasing so the destination buffer edge may not be smooth, especially with a rotation matrix. VGLite path rendering can be used to achieve high-quality coverage sample anti-aliasing (16X, 8X, 4X) rendering effect.

Application can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_DOUBLE_IMAGE) to determine HW support for double image.

Note:

- The vg_lite_blit function can be used for color conversion for Source0 or Source1 before merging sources with vg_lite_blit2.

Syntax:

```
vg_lite_error_t vg_lite_blit2 (
    vg_lite_buffer_t            *target,
    vg_lite_buffer_t            *source0,
    vg_lite_buffer_t            *source1,
    vg_lite_matrix_t            *matrix0,
    vg_lite_matrix_t            *matrix1,
    vg_lite_blend_t             blend,
    vg_lite_filter_t            filter
);
```

Parameters:

Nam	Description
*tar-get	Points to the vg_lite_buffer_t structure, which defines the destination buffer. See Alignment notes for valid destination color formats for the blit functions
*sou *sou	Points to the vg_lite_buffer_t structure for the source0 and source1 buffers. All color formats available in the vg_lite_buffer_format_t‘ enum are valid source formats for the blit functions.
*mat *ma-trix1	Points to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix0 for the source0 pixels and matrix1 for the source1 pixels. If matrix0 and matrix1 are both NULL, the identity matrix is assumed, meaning the blending result of Source0 and Source1 is copied directly on the target at location(0,0).
blen	Specifies one of the enum vg_lite_blend_t values for hardware-supported blend modes to be applied to each image pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0). Note: If the “matrix” parameter is specified with rotation or perspective, and the “blend” parameter is specified as VG_LITE_BLEND_NONE, VG_LITE_BLEND_SRC_IN, or VG_LITE_BLEND_DST_IN, the VGLite driver overwrites the application’s setting for the BLIT operation as follows: - If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is supported, the transparency mode will always be set to TRANSPARENT. - If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is not supported, the blend mode will always be set to VG_LITE_BLEND_SRC_OVER. This is due to some limitations in the VGLite hardware.
fil-ter	Specifies the filter type. All formats available in the vg_lite_filter_t enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit functions

vg_lite_blit_rect **function**

Description:

This is the blit rectangle function. The blit operation is performed using a source and a destination buffer. The source and destination buffer structures are defined using the vg_lite_buffer_t structure. Blit copies a source image to the destination window with a specified matrix that can include translation, rotation, scaling, and perspective correction. Note that vg_lite_buffer_t does not support coverage sample anti-aliasing so the destination buffer edge may not be smooth, especially with a rotation matrix. VGLite path rendering can be used to achieve high-quality coverage sample anti-aliasing (16X, 8X, 4X) rendering effect.

Note:

- The blit_rect function can be used with or without the blend function (vg_lite_blend_t).

- The blit_rect function can be used with or without specifying any color value (vg_lite_color_t).
- The blit_rect function can be used for color conversion with an identity matrix and appropriate formats specified for the source and destination buffers. In this case, do not specify blend mode and color value.
- The vg_lite_blit_rect rectangle start origin point is always (0,0) for hardware versions prior to GCNanoLiteV 1311p that do not support a non-zero rectangle origin.

Syntax:

```
vg_lite_error_t vg_lite_blit_rect (
    vg_lite_buffer_t          *target,
    vg_lite_buffer_t          *source,
    vg_lite_rectangle_t        *rect,
    vg_lite_matrix_t          *matrix,
    vg_lite_blend_t            blend,
    vg_lite_color_t            color,
    vg_lite_filter_t           filter
);
```

Parameters:

Nam	Description
*tar-get	Points to the vg_lite_buffer_t structure that defines the destination buffer.
*sou	Points to the vg_lite_buffer_t structure for the source buffer. All color formats available in the vg_lite_buffer_format_t enum are valid source formats for the blit_rect function.
*rect	Specifies the rectangle area of the source image to blit. rect[0]/[1]/[2]/[3] are x, y, width, and height of the source rectangle respectively. **Note:** Non-zero source origins are supported.
*ma-trix	Points to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of source pixels into the target. If the matrix is NULL, then an identity matrix is assumed, which means that the source is copied directly at 0,0 location on the target.
blen	Specifies one of the enum vg_lite_blend_t values for hardware-supported blend modes to be applied to each image pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0). **Note:** If the matrix parameter is specified with rotation or perspective, and the blend parameter is specified as VG_LITE_BLEND_NONE, VG_LITE_BLEND_SRC_IN, or VG_LITE_BLEND_DST_IN; then, the VGLite driver overwrites the application setting for the blit operation as follows: - If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is supported, then Transparency mode is always set to TRANSPARENT - If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is not supported, then Blend mode is always set to VG_LITE_BLEND_SRC_OVER. It happens due to some limitations in the VGLite hardware.
color	If non-zero, this color value is used as a mix color. The mixed color gets multiplied with each source pixel before blending happens. If you do not need a mix color, then set the color parameter to 0. **Note:** This parameter has no effect if the source vg_lite_buffer_t structure member image_mode is set to VG_LITE_ZERO or VG_LITE_NORMAL_IMAGE_MODE.
fil-ter	Specifies the filter type. All formats available in the vg_lite_filter_t enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit functions

vg_lite_copy_image **function**

Description:

This API copied a pixel rectangle with dimension (width, height) from source buffer to destination buffer. The source image pixel (sx *+ i,*sy + *j*) is copied to the destination image pixel (dx *+ i,*dy + *j*), for *0 □ i <*width and *0 □ j <*height. Pixels whose source or destination lie outside the bounds of the respective image are ignored. Pixel format conversion is applied as needed.

No pre-multiply, transformation, blending, filtering operations are applied to the pixel copy.

Syntax:

```
vg_lite_error_t vg_lite_copy_image (
    vg_lite_buffer_t            *target,
    vg_lite_buffer_t            *source,
    vg_lite_int32_t             sx,
    vg_lite_int32_t             sy,
    vg_lite_int32_t             dx,
    vg_lite_int32_t             dy,
    vg_lite_int32_t             width,
    vg_lite_int32_t             height
);
```

Parameters:

Nam	Description
*tar-get	Points to the vg_lite_buffer_t structure that defines the destination buffer.
*sour	Points to the vg_lite_buffer_t structure for the source buffer. All color formats available in the vg_lite_buffer_format_t enum are valid source formats for the blit function.
sx, sy	Pixel coordinates of the lower-left corner of a pixel rectangle within the source buffer.
dx, dy	Pixel coordinates of the lower-left corner of a pixel rectangle within the target buffer.
widtl	Width of the copied pixel rectangle.
heigh	Height of the copied pixel rectangle.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit functions

vg_lite_get_transform_matrix **function**

Description:

This function generates a 3x3 homogenous transform matrix from 4 float point source coordinates and 4 float point target coordinates. *(from March 2021)*

Syntax:

```
vg_lite_error_t vg_lite_get_transform_matrix (
    vg_lite_float_point4_t      src,
    vg_lite_float_point4_t      dst,
    vg_lite_matrix_t            *mat
);
```

Parameters:

Name	Description
src	Pointer to the four 2D points that form a source polygon
dst	Pointer to the four 2D points that form a destination polygon
mat	Output parameter, pointer to a 3x3 homogenous matrix that transforms the source polygon to a destination polygon.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit functions

vg_lite_clear **function**

Description:

This function performs the clear operation, clearing/filling the specified buffer (entire buffer or partial rectangle in a buffer) with an explicit color.

Syntax:

```
vg_lite_error_t vg_lite_clear (
    vg_lite_buffer_t          *target,
    vg_lite_rectangle_t        *rect,
    vg_lite_color_t           color
);
```

Parameters:

Name	Description
*target	Pointer to the vg_lite_buffer_t structure for the destination buffer. All color formats available in the vg_lite_buffer_format_t enum are valid destination formats for the clear function.
*rect	Pointer to the vg_lite_rectangle_tstructure that specifies the area to be filled. If the rectangle is NULL, the entire target buffer is filled with the specified color.
color	Clear color, as specified in the vg_lite_color_t enum that is the color value to use for filling the buffer. If the buffer is in L8 format, the RGBA color is converted into a luminance value.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit functions

vg_lite_set_color_key **function**

Description:

This function sets a color key. Color key can be used for blit or for draw pattern operations. *(from April 2022)*

A “color key” have two sections, where each section contains R,G,B channels which are noted as high_rgb and low_rgb respectively.

When the vg_lite_color_key_t structure value enable is true, the color key specified is effective and the alpha value is used to replace the alpha channel of the destination pixel when its RGB channels are within range [low_rgb, high_rgb]. After the color key is used in the current frame, if the color key is not needed for the next frame, it should be disabled before the next frame.

Hardware support for color key is not available for GCNanoLiteV. Application can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_COLOR_KEY) to determine HW support for color key.

Syntax:

```
vg_lite_error_t vg_lite_set_color_key (
    vg_lite_color_key4_t          colorkey
);
```

Parameters:

Parameter	Description
colorkey	Color keying parameters as defined by vg_lite_color_key4_t.

Here are 4 groups of color key states:

- color_key_0, high_rgb_0, low_rgb_0, alpha_0, enable_0
- color_key_1, high_rgb_1, low_rgb_1, alpha_1, enable_1
- color_key_2, high_rgb_2, low_rgb_2, alpha_2, enable_2
- color_key_3, high_rgb_3, low_rgb_3, alpha_3, enable_3

The priority order of these states is:

color_key_0 > color_key_1 > color_key_2 > color_key_3.

Returns:

VG_LITE_SUCCESS if successful. VG_LITE_NOT_SUPPORT if color key is not supported in hardware.

Parent topic: Blit functions

vg_lite_gaussian_filter function

Description:

This function sets 3x3 gaussian blur weighted values to filter an image pixel. *(from March 2023)*

The parameters w0, w1, w2 define a 3x3 gaussian blur weight matrix as:

```
|   w2    w1    w2  |
|   w1    w0    w1  |
|   w2    w1    w2  |
```

The sum of the 9 kernel weights must be 1.0 to avoid convolution overflow (w0 + 4*w1 + 4*w2 = 1.0).

The 3x3 weight matrix applies to a 3x3 pixel block:

```
| pixel[i-1][j-1]     pixel[i][j-1]       pixel[i+1][j-1]|
| pixel[i-1][j]       pixel[i][j]         pixel[i+1][j]  |
| pixel[i-1][j+1]     pixel[i][j+1]       pixel[i+1][j+1]|
```

With the following dot product equation:

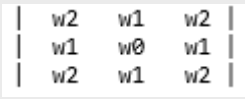
```
color[i][j] = w2*pixel[i-1][j-1] + w1*pixel[i][j-1] + w2*pixel[i+1][j-1]
            + w1*pixel[i-1][j]   + w0*pixel[i][j]   + w1*pixel[i+1][j]
            + w2*pixel[i-1][j+1] + w1*pixel[i][j+1] + w2*pixel[i+1][j+1];
```

Applications can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_GAUSSIAN_BLUR) to determine HW support for gaussian blur.

Syntax:

```
vg_lite_error_t vg_lite_gaussian_filter (
    vg_lite_float_t             w0
    vg_lite_float_t             w1
    vg_lite_float_t             w2
);
```

Parameters:

Parameter	Description
w0, w1, w2	w0, w1, w2 define a 3x3 gaussian blur weighted matrix as: 

Returns:

VG_LITE_SUCCESS if successful. Otherwise, VG_LITE_NOT_SUPPORT if gaussian blur is not supported in hardware.

Parent topic: Blit functions

Parent topic:*Blits for compositing and blending*

Blit/Draw extended functions

The following BLIT or DRAW-related functions typically require GC355 or GC555 hardware and are not available for all Vivante Vector Graphics hardware configurations.

Applications can use the VGLite API vg_lite_query_feature to determine HW support for the related functionality.

vg_lite_get_parameter function

Description:

This function returns the selected VGLite / GPU states to the application.

(from Aug 2023)

Syntax:

```
vg_lite_error_t vg_lite_get_parameter (
    vg_lite_param_type_t        type,
    vg_lite_int32_t             count,
    vg_lite_pointer             params
);
```

Parameters:

Parameter	Description
type	The parameter type to be queried (VG_LITE_GPU_IDLE_STATE, VG_LITE_SCISSOR_RECT)
count	The number of returned parameters
params	The pointer to the array of returned parameters

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_enable_scissor function

Description:

This function enables scissor rectangle operation for the rectangle regions defined by vg_lite_scissor_rects API. *(from March 2020, modified August 2020, requires GC355 or GC555 hardware)*

Applications can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_SCISSOR) to determine HW support for scissoring. Support is available with GC355 and GC555.

Syntax:

```
vg_lite_error_t vg_lite_enable_scissor (
    void
);
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_disable_scissor function

Description:

This function disables scissor operation for the rectangle regions defined by the vg_lite_scissor_rects API. *(from March 2020, modified August 2020, requires GC355 or GC555 hardware)*.

Syntax:

```
vg_lite_error_t vg_lite_disable_scissor (
    void
);
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_scissor_rects function

Description:

This function defines scissor rectangle regions on the hardware mask layer. But the scissor function is enable/disabled by vg_lite_enable_scissor and vg_lite_disable_scissor APIs. *(from August 2022, requires GC355 or GC555 hardware).*

Syntax:

```
vg_lite_error_t vg_lite_scissor_rects (
    vg_lite_buffer_t            *target,
    vg_lite_uint32_t             nums,
    vg_lite_rectangle_t          rect[]
);
```

Parameters:

Parameter	Description
target	Target render buffer that has the scissor mask layer.
nums	Number of scissor rectangles.
rect[]	The scissor rectangle array.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_set_scissor function

Description:

This is a legacy scissor API function that can be used to set a single scissor rectangle for the render target. This scissor API is supported by a different hardware mechanism other than the mask layer and it has better performance than the mask layer scissor function.

This API is not enabled/disabled by vg_lite_enable_scissor and vg_lite_disable_scissor APIs. Instead, the vg_lite_set_scissor API calls with a valid scissor rectangle input (x, y, right, bottom) enables the scissor function by default. The vg_lite_set_scissor API call with input parameter (-1, -1, -1, -1) disables the scissor function. *(requires GC355 or GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_set_scissor (
    vg_lite_int32_t             x,
    vg_lite_int32_t             y,
    vg_lite_int32_t             right,
    vg_lite_int32_t             bottom
);
```

Parameters:

Parameter	Description
x	X Origin of rectangle, left coordinate in pixels
Y	Y Origin of rectangle, top coordinate in pixels
right	X rightmost pixel of the rectangle
bottom	Y bottom pixel of the rectangle

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_disable_color_transform **function**

Description:

This function is used to disable color transformation. By default, the color transform is turned off. *(from Sept 2022, only for GC355 and GC555 hardware)*

Applications can use the VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_COLOR_TRANSFORMATION)‘ to determine HW support for color transformation. Support is available with GC355 and GC555.

Syntax:

```
vg_lite_error_t vg_lite_disable_color_transform (
);
```

Parameters: None

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_enable_color_transform **function**

Description:

This function is used to enable color transformation. By default, the color transform is turned off. *(from Sept 2022, only for GC355 and GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_enable_color_transform (
);
```

Parameters: None

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_set_color_transform **function**

Description:

This function is used to set pixel scale and bias values for color transformation for each pixel channel. *(from August 2022, only for GC355 and GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_set_color_transform (
    vg_lite_color_transform_t      *values
);
```

Parameters:

Parameter	Description
*values	Pointer to the color transformation values to set. See enum vg_lite_color_transform_t.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_enable_masklayer **function**

Description:

This function controls the availability of mask functionality. The mask is turned off by default. *(from August - Sept mber 2022, requires GC555 hardware)*

Applications can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_MASK) to determine HW support for mask. The blit and draw mask functions below require GC555 hardware support. These functions were introduced in August 2022 and the syntax or name was further refined in September 2022.

Syntax:

```
vg_lite_error_t vg_lite_enable_masklayer (
    void
  );
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_disable_masklayer **function**

Description:

This function controls the availability of mask functionality. The mask is turned off by default. *(from August -September 2022, requires GC555 hardware, prior to Sept 2022 name was vg_lite_disable_mask_layer)*

Syntax:

```
vg_lite_error_t vg_lite_disable_masklayer (
    void
);
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_create_masklayer **function**

Description:

This function creates a mask layer with the specified width and height. The mask format defaults to A8 and the default mask value is 255. *(from August 2022-September, requires GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_create_masklayer (
    vg_lite_buffer_t            *masklayer,
    vg_lite_uint32_t             width,
    vg_lite_uint32_t             height
);
```

Parameters:

Parameter	Description
*masklayer	Points to the address of the buffer of the mask layer to be created.
width	Mask layer width (in pixels).
height	Mask layer height (in pixels).

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_blend_masklayer **function**

Description:

This function blends the specified area of the source mask layer with the destination mask layer according to an vg_lite_mask_operation_t enumeration value, to create a blended destination mask layer. *(from August-September 2022, requires GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_blend_masklayer (
    vg_lite_buffer_t            *dst_masklayer,
    vg_lite_buffer_t            *src_masklayer,
    vg_lite_mask_operation       operation,
    vg_lite_rectangle_t          *rect,
);
```

Parameters:

Parameter	Description
*dst_masklay	Points to the address of the buffer of the destination mask layer.
*src_masklay	Points to the address of the buffer of the source mask layer.
operation	Blending mode to be applied to each image pixel, as defined by the enum vg_lite_mask_operation_t.
*rect	The rectangle area (x, y, width, height) of the blend operation.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_set_masklayer **function**

Description:

This function sets the given mask layer to the hardware. *(from August-September 2022, requires GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_set_masklayer (
    vg_lite_buffer_t              *masklayer
);
```

Parameters:

Parameter	Description
*masklayer	Points to the address of the buffer of the mask layer to be set.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_render_masklayer **function**

Description:

This function draws the mask layer according to the specified path, color, and matrix information. *(from August-September 2022, requires GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_render_masklayer (
    vg_lite_buffer_t              *masklayer,
    vg_lite_mask_operation         operation,
    vg_lite_path_t                *path,
    vg_lite_fill_t                fill_rule,
    vg_lite_color_t               color,
    vg_lite_matrix_t              *matrix
);
```

Parameters:

Parameter	Description
*mas	Points to the address of the buffer of the destination mask layer.
operation	Blending mode to be applied to each image pixel, as defined by the enum vg_lite_mask_operation_t
*patl	Pointer to the vg_lite_path_t structure containing path data that describes the path to draw. Refer to Vector path opcodes for plotting paths in this document for opcode detail.
fill_r	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
color	Specifies the color vg_lite_color_t RGBA value to be applied to each pixel drawn by the path.
*matrix	Points to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If the matrix is NULL, an identity matrix is assumed, meaning the source is copied directly on the target at 0,0 location. which is usually a bad idea since the path can be anything.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_destroy_masklayer **function**

Description:

This function is used to free a mask layer. *(from August-September 2022, requires GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_destroy_masklayer (
    vg_lite_buffer_t              masklayer
);
```

Parameters:

Parameter	Description
*masklayer	Points to the address of the buffer of the mask layer to be destroyed.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_set_mirror **function**

Description:

This function is used to control mirror functionality. By default, the mirror is turned off and the default output orientation is from top to bottom. *(from August 2022, only for GC555 hardware)*

Application can use VGLite API [vg_lite_query_feature](vg_lite_query_feature_function.md) (gcFEATURE_BIT_VG_MIRROR) to determine HW support for mirror. Mirror functions require GC555 hardware.

Syntax:

```
vg_lite_error_t vg_lite_set_mirror (
    vg_lite_orientation_t         orientation
);
```

Parameters:

Parameter	Description
orientation	The orientation mode as defined by the enum vg_lite_orientation_t.‘

Returns:

VG_LITE_SUCCESS or VG_LITE_NOT_SUPPORT if not supported.

Parent topic: Blit/Draw extended functions

vg_lite_source_global_alpha **function**

Description:

This function sets the image/source global alpha and return a status error code. *(from June 2021, requires GCNanoUltraV or GC555 hardware)*

Application can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_GLOBAL_ALPHA) to determine HW support for global alpha. The global alpha BLIT-related functions require GCNanoUltraV or GC555 hardware.

Syntax:

```
vg_lite_error_t vg_lite_source_global_alpha (
    vg_lite_global_alpha_t        alpha_mode,
    vg_lite_uint8_t               alpha_value
);
```

Parameters:

Parameter	Description
alpha_mode	Global alpha mode value. See enum vg_lite_global_alpha_t.
alpha_value	The image/source global alpha value to set.

Returns:

VG_LITE_SUCCESS or VG_LITE_NOT_SUPPORT if global alpha is not supported.

Parent topic: Blit/Draw extended functions

vg_lite_dest_global_alpha **function**

Description:

This function sets the destination global alpha and returns a status error code. *(from June 2021, requires GCNanoUltraV or GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_dest_global_alpha (
    vg_lite_global_alpha_t        alpha_mode,
    vg_lite_uint8_t               alpha_value
);
```

Parameters:

Parameter	Description
alpha_mode	Global alpha mode value. See enum vg_lite_global_alpha_t.
alpha_value	The destination global alpha value to set.

Returns:

VG_LITE_SUCCESS or VG_LITE_NOT_SUPPORT if global alpha is not supported.

Parent topic: Blit/Draw extended functions

Parent topic:*Blits for compositing and blending*

Vector path control This chapter provides an overview of the vector path enumerations, structures, functions, and opcodes for plotting paths.

Vector path enumerations

This section provides an overview of vector path enumerations.

vg_lite_format_t enumeration

Values for vg_lite_format_t enum are defined in Table 1.

If vg_lite_format_t	Path data alignment in the array should be:
VG_LITE_S8	8 bit
VG_LITE_S16	2 bytes
VG_LITE_S32	4 bytes

Parent topic: Vector path enumerations

vg_lite_quality_t enumeration

Specifies the level of hardware assisted anti-aliasing.

Used in structure: vg_lite_path_t.

Used in function: vg_lite_init_path, vg_lite_init_arc_path.

vg_lite_qua string values	Description
VG_LITE_	High quality: 16x coverage sample anti-aliasing
VG_LITE_	Upper quality: 8x coverage sample anti-aliasing. Use vg_lite_query_feature to determine availability of 8x CSAA (feature enum value gcFEATURE_BIT_VG_QUALITY_8X.(deprecated from June 2020, available with supported hardware from August 2022).
VG_LITE_	Medium quality: 4x coverage sample anti-aliasing
VG_LITE_	Low quality: No anti-aliasing

Parent topic: Vector path enumerations

Parent topic:*Vector path control*

Vector path structures

This section provides an overview of vector path structures.

vg_lite_hw_memory structure

This structure gets the memory allocation information recorded by the kernel.

Used in structure: vg_lite_path_t.

vg_lite_hw_mem member	Type	Description
handle	vg_lite_p	GPU memory object handle
memory	vg_lite_p	Logical memory address
address	vg_lite_u	GPU memory address
bytes	vg_lite_u	Size of memory
property	vg_lite_u	Bit 0 is used for path upload: - 0: Disable path data uploading (always embedded into command buffer) - 1: Enable auto path data uploading

Parent topic: Vector path structures

vg_lite_path_t **structure**

This structure describes VGLite path data.

Path data is made of op codes and coordinates. The format for op codes is always VG_LITE_S8. For more details on opcodes, see Vector path opcodes for plotting paths.

Used in init functions: vg_lite_init_path, vg_lite_init_arc_path, vg_lite_upload_path, vg_lite_clear_path, vg_lite_append_path.

Used in draw functions: vg_lite_draw, vg_lite_draw_grad, vg_lite_draw_radial_grad, vg_lite_draw_pattern.

vg_lite_path_t members	Type	Description
bounding_box[4]	vg_lite_float_	bounding box for path [0] left [1] top [2] right [3] bottom
quality	vg_lite_quality	enum for quality hint for the path, anti-aliasing level
format	vg_lite_format	enum for coordinate format
uploaded	vg_lite_hw_me	struct with path data that has been uploaded into GPU addressable memory
path_length	vg_lite_uint32	number of bytes in the path
path	vg_lite_pointe	pointer to path data
path_changed	vg_lite_int8_t	0: not changed; 1: changed.
pdata_internal	vg_lite_int8_t	0: path data memory is allocated by application; 1: path data memory is allocated by driver.
path_type	vg_lite_path_t	The draw path type as specified in enum vg_lite_path_type_t. *(added for stroke control, from March 2022)*
*stroke	vg_lite_stroke	As defined by structure vg_lite_stroke_t *(added for stroke control, from March 2022)*
stroke_path	vg_lite_pointe	Pointer to the physical description of the stroke path. *(added for stroke control, from March 2022)*
stroke_size	vg_lite_uint32	Number of bytes in the stroke path data. *(added for stroke control, from March 2022)*
stroke_color	vg_lite_color_t	The stroke path fill color. *(from Sept 2022)*
add_end	vg_lite_int8_t	Flag that add end_path in driver *(from March 2023)*

Special notes for path objects:

- Endianness has no impact, as it is aligned against the boundaries
- Multiple contiguous opcodes should be packed by the size of the specified data format. For example, by 2 bytes for VG_LITE_S16 or by 4 bytes for VG_LITE_S32.

For example, because opcodes are 8-bit (1-byte), 16-bit (2-byte), or 32-bit (4-byte) data types:

```
...
<opcode1_that_needs_data>
<align_to_data_size>
<data_for_opcode1>
<opcode2_that_doesnt_need_data>
<align_to_data_size>
<opcode3_that_needs_data>
<align_to_data_size>
<data_for_opcode3>
...
```

- Path data in the array should always be 1-, 2-, or 4-byte aligned, depending on the format:

For example, for 32-bit (4-byte) data types:

```
...
<opcode1_that_needs_data>
<pad to 4 bytes>
<4 byte data_for_opcode1>
<opcode2_that_doesnt_need_data>
<pad to 4 bytes>
<opcode3_that_needs_data>
<pad to 4 bytes>
<4 byte data_for_opcode3>
...
```

Parent topic: Vector path structures

Parent topic:*Vector path control*

Vector path functions

When using a small tessellation window and depending on a path's size, a path might be uploaded to the hardware multiple times because the hardware scanline convert path with the provided tessellation window size, so VGLite path rendering performance might go down. That is why it is preferable to set the tessellation buffer size to the most common path size, for example if you only render 24-pt fonts, you can set the tessellation buffer to be 24x24.

All the RGBA color formats available in the vg_lite_buffer_format_t are supported as the destination buffer for the draw function.

vg_lite_get_path_length **function**

Description:

This function calculates the path command buffer length (in bytes).

The application is responsible for allocating a buffer according to the buffer length calculated with this function. Then, the buffer is used by the path as a command buffer. The VGLite driver does not allocate the path command buffer.

Syntax:

```
vg_lite_uint32_t vg_lite_get_path_length (
    vg_lite_uint8_t             *opcode,
    vg_lite_uint32_t            count,
    vg_lite_format_t            format
);
```

Parameters:

Parameter	Description
*opcode	Pointer to the opcode array to use to construct the path. *(*opcode from March 2023)*
count	The opcode count
format	The coordinate data format. All formats available in the vg_lite_format_t enum are valid formats for this function.

Returns:

Returns the command buffer length in bytes.

Parent topic: Vector path functions

vg_lite_append_path function

Description:

This function assembles the command buffer for the path. The command buffer is allocated by the application and assigned to the path. This function makes the final GPU command buffer for the path based on the input opcodes (cmd) and coordinates (data). The application is responsible for allocating a buffer large enough for the path*. (from Jan 2022, returns a vg_lite_error_t status code)*

Syntax:

```
vg_lite_error_t vg_lite_append_path (
    vg_lite_path_t          *path
    vg_lite_uint8_t         *opcode,
    vg_lite_pointer         data,
    vg_lite_uint32_t        seg_count
);
```

Parameters:

Parameter	Description
*path	Pointer to the vg_lite_path_t structure with the path definition.
*opcode	Pointer to the opcode array to use to construct the path. *(*opcode from March 2023)*
data	Pointer to the coordinate data array to use to construct the path
seg_count	The opcode count

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Vector path functions

vg_lite_init_path function

Description:

This function initializes a path definition with specified values. *(From Dec 2019 returns vg_lite_error_t, previous was void.)*

Syntax:

```
vg_lite_error_t vg_lite_init_path (
    vg_lite_path_t              *path,
    vg_lite_format_t             format,
    vg_lite_quality_t           quality,
    vg_lite_uint32_t            length,
    vg_lite_pointer             *data,
    vg_lite_float_t             min_x,
    vg_lite_float_t             min_y,
    vg_lite_float_t             max_x,
    vg_lite_float_t             max_y
);
```

Parameters:

Parameter	Description
*path	Pointer to the vg_lite_path_t structure for the path object to be initialized with the member values specified.
format	The coordinate data format. All formats available in the vg_lite_format_t enum are valid formats for this function.
quality	The quality for the path object. All formats available in the vg_lite_quality_t enum are valid formats for this function.
length	The length of the path data (in bytes)
*data	Pointer to path data
min_x min_y max_x max_y	Minimum and maximum x and y values specifying the bounding box of the path

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Vector path functions

vg_lite_init_arc_path **function**

Description:

This function initializes an arc path definition with specified values. *(from February 2021)*

Syntax:

```
vg_lite_error_t vg_lite_init_arc_path (
    vg_lite_path_t              *path,
    vg_lite_format_t             format,
    vg_lite_quality_t           quality,
    vg_lite_uint32_t            length,
    vg_lite_pointer             *data,
    vg_lite_float_t             min_x,
    vg_lite_float_t             min_y,
    vg_lite_float_t             max_x,
    vg_lite_float_t             max_y
);
```

Parameters:

Parameter	Function
*path	Pointer to the vg_lite_path_t structure for the path object to be initialized with the member values specified.
format	The coordinate data format. The vg_lite_format_t enum value should be FP32.
quality	The quality for the path object. All formats available in the vg_lite_quality_t enum are valid formats for this function.
length	The length of the path data (in bytes).
*data	Pointer to path data.
min_x min_y max_x max_y	Minimum and maximum x and y values specifying the bounding box of the path.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Vector path functions

vg_lite_upload_path **function**

Description:

This function is used to upload a path to GPU memory.

In normal cases, the VGLite driver will copy any path data into a command buffer structure during runtime. This does take some time if there are many paths to be rendered. Also, in an embedded system the path data won't change - so it makes sense to upload the path data into GPU memory in such a form that the GPU can directly access it. This function will signal the driver to allocate a buffer that will contain the path data and the required command buffer header and footer data for the GPU to access the data directly. Call vg_lite_clear_path to free this buffer after the path is used.

Syntax:

```
vg_lite_error_t vg_lite_upload_path (
    vg_lite_path_t              *path
);
```

Parameters:

Parameter	Description
*path	Pointer to a vg_lite_path_t structure that contains the path to be uploaded.

Returns:

VG_LITE_OUT_OF_MEMORY if not enough GPU memory is available for buffer allocation.

Parent topic: Vector path functions

vg_lite_clear_path **function**

Description:

This function will clear and reset path member values. If the path has been uploaded, it frees the GPU memory allocated when uploading the path. *(From Dec 2019 returns vg_lite_error_t, previous was void.)*

.

Syntax:

```
vg_lite_error_t vg_lite_clear_path (
    vg_lite_path_t              *path
);
```

Parameters:

Parameter	Description
*path	Pointer to the vg_lite_path_t path definition to be cleared.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Vector path functions

Parent topic:*Vector path control*

Vector path opcodes for plotting paths

The following opcodes are path drawing commands available for vector path data.

A path operation is submitted to the GPU as [Opcode | Coordinates]. The operation code is stored as a VG_LITE_S8 while the coordinates are specified via vg_lite_format_t.

Op-code	Arguments	Description
0x00	None	**VLC_OP_END**. Finish tessellation. Close any open path.
0x01	None	**VLC_OP_CLOSE**. For VGLite driver internal use only. Application should not use this OP directly.
0x02	(x, y)	**VLC_OP_MOVE**. Move to the given vertex. Close any open path. $start_x = x$ $start_y = y$
0x03	(□x, □y)	**VLC_OP_MOVE_REL**. Move to the given relative point. Close any open path. $start_x = start_x + \Delta x$ $start_y = start_y + \Delta y$
0x04	(x, y)	**VLC_OP_LINE**. Draw a line to the given point. $Line(start_x, start_y, x, y)$ $start_x = x$ $start_y = y$
0x05	(□x, □y)	**VLC_OP_LINE_REL**. Draw a line to the given relative point. $x = start_x + \Delta x$ $y = start_y + \Delta y$ $Line(start_x, start_y, x, y)$ $start_x = x$ $start_y = y$
0x06	(cx, cy) (x, y)	**VLC_OP_QUAD**. Draw a quadratic Bezier curve to the given end point using the specified control point. $Quad(start_x, start_y, cx, cy, x, y)$ $start_x = x$ $start_y = y$
0x07	(□cx, □cy) (□x, □y)	**VLC_OP_QUAD_REL**. Draw a quadratic Bezier curve to the given relative end point using the specified relative control point. $cx = start_x + \Delta cx$ $cy = start_y + \Delta cy$ $x = start_x + \Delta x$ $y = start_y + \Delta y$ $Quad(start_x, start_y, cx, cy, x, y)$ $start_x = x$ $start_y = y$
		VLC_OP_CUBIC. Draw a cubic Bezier curve to the given end point using the specified control points. $Cubic(start_x, start_y, cx_1, cy_1, cx_2, cy_2, x, y)$ $start_x = x$

Parent topic:*Vector path control*

Vector-dased draw operations This part of the API performs the hardware accelerated draw operations.

Draw and gradient enumerations

This section provides an overview of draw and gradient enumerations.

vg_lite_blend_t enumeration

This enumeration is defined under the “Blit enumerations” section (see vg_lite_blend_t enumeration).

Parent topic: Draw and gradient enumerations

vg_lite_color_t parameter

The common parameter vg_lite_color_t is described in Common parameter types.

Parent topic: Draw and gradient enumerations

vg_lite_fill_t enumeration

This enumeration is used to specify the fill rule to use. For drawing any path, the hardware supports both non-zero and odd-even fill rules.

To determine whether any point is contained inside an object, imagine drawing a line from that point out to infinity in any direction such that the line does not cross any vertex of the path. For each edge that is crossed by the line, add 1 to the counter if the edge is crossed from left to right, as seen by an observer walking across the line towards infinity, and subtract 1 if the edge crossed from right to left. In this way, each region of the plane will receive an integer value.

The non-zero fill rule says that a point is inside the shape if the resulting sum is not equal to zero. The even/odd rule says that a point is inside the shape if the resulting sum is odd, regardless of sign.

Used in function: vg_lite_render_masklayer.

Used in draw functions: vg_lite_draw, vg_lite_draw_grad, vg_lite_draw_radial_grad, vg_lite_draw_pattern.

vg_lite_fill_t string values	Description
VG_LITE_FILL_NON_ZEI	Non-zero fill rule. A pixel is drawn if it crosses at least one path pixel.
VG_LITE_FILL_EVEN_OI	Even-odd fill rule. A pixel is drawn if it crosses an odd number of path pixels.

Parent topic: Draw and gradient enumerations

vg_lite_filter_t enumeration

This enum is defined under the “Blit enumerations” section (see vg_lite_filter_t enumeration).

Parent topic: Draw and gradient enumerations

vg_lite_gradient_spreadmode_t enumeration

vg_lite_gradient_spreadmode_t enum is defined to match OpenVG enum VGColorRampSpreadMode *(from March 2023, replaces*vg_lite_radial_gradient_spreadmode*, requires GC355/GC555 hardware)**

The application may only define stops with offsets between 0 and 1. Spread modes define how the given set of stops are repeated or extended in order to define interpolated color values for arbitrary input values outside the [0,1] range.

Used in structure: vg_lite_radial_gradient_t.

vg_lite_gradient_spreadı String Values	Description
VG_LITE_GRADIENT_S	The current fill color is used for all stop values less than 0 or greater than 1 respectively.
VG_LITE_GRADIENT_S	Colors defined at 0 and 1 are used for all stop values less than 0 or greater than 1 respectively.
VG_LITE_GRADIENT_S	Color values defined between 0 and 1 are repeated indefinitely in both directions.
VG_LITE_GRADIENT_S	Color values defined between 0 and 1 are repeated indefinitely in both directions but with alternate copies of the range reversed.

Parent topic: Draw and gradient enumerations

vg_lite_pattern_mode_t **enumeration**

Defines how the region outside the image pattern is filled for the path.

Used in function: vg_lite_draw_gradient, vg_lite_draw_pattern.

vg_lite_pattern_ string values	Description
VG_LITE_PAT	Pixels outside the bounds of the source image should be taken as the color.
VG_LITE_PAT	Pixels outside the bounds of the source image should be taken as having the same color as the closest edge pixel. The color of the pattern border is expanded to fill the region outside the pattern.
VG_LITE_PAT	Pixels outside the bounds of the source image should be repeated indefinitely in all directions. *(from March 2023)*
VG_LITE_PAT	Pixels outside the bounds of the source image should be reflected indefinitely in all directions. *(from March 2023)*

Parent topic: Draw and gradient enumerations

vg_lite_radial_gradient_spreadmode_t **enumeration**

(Deprecated March 2023) use vg_lite_gradient_spreadmode_t. Defines the radial gradient padding mode. *(from Nov 2020, requires GC355 hardware)*

Used in structure: vg_lite_radial_gradient_t.

vg_lite_radial_gradient_spread String Values	Description
VG_LITE_RADIAL_GRADI = 0	The current fill color is used for all stop values less than 0 or greater than 1 respectively.
VG_LITE_RADIAL_GRADI	Colors defined at 0 and 1 are used for all stop values less than 0 or greater than 1 respectively.
VG_LITE_RADIAL_GRADI	Color values defined between 0 and 1 are repeated indefinitely in both directions.
VG_LITE_RADIAL_GRADI	Color values defined between 0 and 1 are repeated indefinitely in both directions but with alternate copies of the range reversed.

Parent topic: Draw and gradient enumerations

Parent topic:*Vector-dased draw operations*

Draw and gradient structures

This section provides an overview of the draw and gradient structures.

vg_lite_buffer_t structure

This structure is defined under the “Pixel buffer structures” section (see vg_lite_buffer_t structure).

Parent topic: Draw and gradient structures

vg_lite_color_ramp_t structure

This structure defines the stops for the radial gradient. The five parameters provide the offset and color for the stop. Each stop is defined by a set of floating point values which specify the offset and the sRGBA color and alpha values. Color channel values are in the form of a non-premultiplied (R, G, B, alpha) quad. All parameters are in the range of [0,1]. The red, green, blue, alpha value of [0, 1] is mapped to an 8-bit pixel value [0, 255].*(from November 2020, requires GC355 hardware)*

The define for the max number of radial gradient stops is #define MAX_COLOR_RAMP_STOPS256.

Used in radial gradient structure: vg_lite_radial_gradient_t.

vg_lite_color_ramp_t members	Type	Description
stop	vg_lite_float_t	Offset value for the color stop
red	vg_lite_float_t	Red color channel value for the color stop
green	vg_lite_float_t	Green color channel value for the color stop
blue	vg_lite_float_t	Blue color channel value for the color stop
alpha	vg_lite_float_t	Alpha color channel value for the color stop

Parent topic: Draw and gradient structures

vg_lite_linear_gradient_t structure

This structure defines the organization of a linear gradient in VGLite data. The linear gradient is applied to filling a path. It generates a 256x1 image according to the specified settings.

Used in init and draw functions: vg_lite_init_grad, vg_lite_set_grad, vg_lite_update_grad, vg_lite_get_grad_matrix, vg_lite_clear_grad, vg_lite_draw_grad.

vg_lite_linear_gradient_t constants	Type	Description
VLC_MAX_GRADIENT_STOP	vg_lite_int32_	Constant. Maximum number of gradient colors = 16.
vg_lite_linear_gradient_t members		
colors [VLC_MAX_GRADIENT_STOF	vg_lite_uint32	Color array for the gradient
count	vg_lite_uint32	Number of colors
stops [VLC_MAX_GRADIENT_STOF	vg_lite_uint32	Number of color stops, from 0 to 255
matrix	vg_lite_matrix	Struct for the matrix to transform the gradient color ramp
image	vg_lite_buffer_	Image object struct to represent the color ramp

Parent topic: Draw and gradient structures

vg_lite_ext_linear_gradient **structure**

This structure defines the organization of the extended parameters possible for a linear gradient *(from April 2022)*.

Used in functions: vg_lite_draw_linear_grad.

vg_lite_ext_linear_gradient members	Type	Description
count	vg_lite_uint32_t	Count of colors, up to 256.
matrix	vg_lite_matrix_t	The matrix to transform the gradient.
image	vg_lite_buffer_t	The image for rendering as gradient pattern.
linear_grad	vg_lite_linear_gra	Linear gradient parameters. Includes center point, focal point and radius.
ramp_length	vg_lite_uint32_t	Color ramp length for gradient paints provided to the driver.
color_ramp[VLC_MAX_C(	vg_lite_color_ram	Color ramp parameter for gradient paints provided to the driver.
converted_length	vg_lite_uint32_t	Converted internal color ramp length.
converted_ramp[VLC_MAX_(	vg_lite_color_ram	Converted internal color ramp.
pre-multiplied	vg_lite_uint8_t	If this value is set to 1, the color value of color_ramp will be multiplied by the alpha value of color_ramp.
spread_mode	vg_lite_radial_gra	The spread mode that is applied to the pixels out of the image after transformed.

Parent topic: Draw and gradient structures

vg_lite_linear_gradient_parameter **structure**

This structure defines a radial direction for a linear gradient. *(from April 2022)*

Line0 connects point (X0, Y0) to point (X1, Y1) and represents the radial direction of the linear gradient.

Line1 is a line perpendicular to line0 which passes through point (X0, Y0).

Line2 is a line perpendicular to line0 which passes through point (X1, Y1)

The linear gradient paint is applied at the intersection of the path fill area and the plane starting from line 1 and ending at line 2.

Used in structure: vg_lite_ext_linear_gradient.

Used in functions: vg_lite_set_linear_grad.

vg_lite_linear_gradient_parameter_t members	Type	Description
X0	vg_lite_float_	X origin of linear gradient radial direction.
Y0	vg_lite_float_	Y origin of linear gradient radial direction.
X1	vg_lite_float_	X end point of linear gradient radial direction.
Y1	vg_lite_float_	Y end point of linear gradient radial direction.

Parent topic: Draw and gradient structures

vg_lite_matrix_t **structure**

This structure is defined under the “Matrix control structures” section (see vg_lite_matrix_t structure).

Parent topic: Draw and gradient structures

vg_lite_path_t **structure**

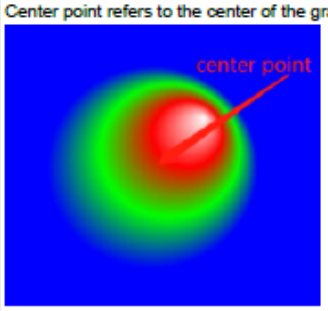
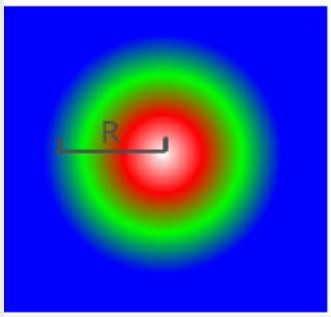
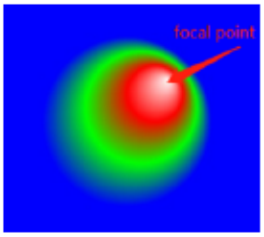
This structure is defined under the “Vector path structures” section (see vg_lite_path_t structure).

Parent topic: Draw and gradient structures

vg_lite_radial_gradient_parameter_t **structure**

This structure defines the gradient radius and the X and Y coordinates for the center and focal points of the gradient *(from November 2020, requires GC355 or GC555 hardware)*.

Used in radial gradient structure: vg_lite_radial_gradient_t.

vg_lite_radial_gradient_parameter_t member	Type	Description
cx	vg_lite_float_t	Coordinates x and y of the gradient color center point. Center point refers to the center of the gradient color. 
cy	vg_lite_float_t	
r	vg_lite_float_t	Radius of the gradient 
fx	vg_lite_float_t	Coordinates x and y of the gradient color focal point. Focal point refers to the center of the gradient color. 
fy	vg_lite_float_t	

Parent topic: Draw and gradient structures

vg_lite_radial_gradient_t **structure**

This structure defines the application of the radial gradient to fill a path. *(from November 2020, requires GC355 or GC555 hardware).*

Used in radial gradient functions: vg_lite_draw_grad, vg_lite_set_radial_grad, vg_lite_update_radial_grad, vg_lite_get_radial_grad, vg_lite_clear_radial_grad.

vg_lite_radial_gradient_t member	Type	Description
count	vg_lite_uint32_t	Count of colors, up to 256
matrix	vg_lite_matrix_t	Structure that specifies the transform matrix for the gradient
image	vg_lite_buffer_t	Structure that specifies the image for rendering as a gradient pattern
radial_grad	vg_lite_radial_gra	Structure that specifies the location of the gradient's center point (cx, cy), focal point(fx, fy) and radius(r)
ramp_length	vg_lite_uint32_t	Color ramp parameters for gradient paints provided to the driver
color_ramp[VLC_MAX_C	vg_lite_color_ram	Structure that specifies the color ramp
converted_length	vg_lite_uint32_t	Converted internal color ramp.
converted_ramp[VLC_MAX_(	vg_lite_color_ram	Structure that specifies the internal color ramp
pre_multiplied	vg_lite_uint32_t	If this value is set to 1, the color value of color_ramp will be multiplied by the alpha value of color_ramp.
spread_mode	vg_lite_radial_gra	Enum that specifies the tiling mode, which is applied to the pixels out of the image after transformation

Parent topic: Draw and gradient structures

Parent topic:*Vector-dased draw operations*

Draw functions

This section provides an overview of the draw functions.

vg_lite_draw function

Description:

This function performs a hardware accelerated 2D vector draw operation.

The size of the tessellation buffer can be specified at initialization and it is aligned with the minimum hardware alignment requirements of the kernel. Specifying a smaller size for tessellation buffer allocates less memory but reduces performance. Because the hardware walks the target with the provided tessellation window size, a path may be sent to the hardware multiple times. It is a good practice to set the tessellation buffer size to the most common path size. For example, if all you do is render up to 24-point fonts, you can set the tessellation buffer to 24x24.

Note:

- All the color formats available in the vg_lite_buffer_format_t enum are supported as the destination buffer for the draw function
- The hardware does not support strokes; they must be converted to paths before you use them in the draw API

Syntax:

```
vg_lite_error_t vg_lite_draw (
    vg_lite_buffer_t        *target,
    vg_lite_path_t          *path,
    vg_lite_fill_t          fill_rule,
    vg_lite_matrix_t        *matrix,
    vg_lite_blend_t         blend,
```

(continues on next page)

(continued from previous page)

```
    vg_lite_color_t          color
);
```

Parameters:

Parameter	Description
*target	Pointer to the vg_lite_buffer_t structure for the destination buffer. All color formats available in the vg_lite_buffer_format_t enum are valid destination formats for the draw function.
*path	Pointer to the vg_lite_path_t structure containing path data that describes the path to draw. See opcode details in Vector path opcodes for plotting paths.
fill_r	Specifies the vg_lite_fill_t enum value for the fill rule for the path
*matrix	Pointer to a vg_lite_matrix_t structure that defines the *affine* transformation matrix of the path. If the matrix is NULL, an identity matrix is assumed. **Note:** Non-affine transformations are not supported by vg_lite_draw; therefore, a perspective transformation matrix might have unexpected effects on path rendering.
blend	Select one of the hardware-supported blend modes in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
color	The color applied to each pixel drawn by the path.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Draw functions

vg_lite_draw_grad function

Description:

This function is used to fill a path with a linear gradient according to the specified fill rules. The specified path is transformed according to the selected matrix and is filled with the specified color gradient.

Syntax:

```
vg_lite_error_t vg_lite_draw_grad (
    vg_lite_buffer_t            *target,
    vg_lite_path_t              *path,
    vg_lite_fill_t              fill_rule,
    vg_lite_matrix_t            *matrix,
    vg_lite_linear_gradient_t   *grad,
    vg_lite_blend_t             blend
);
```

Parameters:

Parameter	Description
*target	Pointer to the vg_lite_buffer_t structure containing data describing the target path.
*path	Pointer to the vg_lite_path_t structure containing path data that describes the path to draw and fill with the linear gradient. See opcode details in Vector path opcodes for plotting paths.
fill_ru	Specifies the vg_lite_fill_t enum value for the fill rule for the path
*matrix	Pointer to the vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If the matrix is NULL, an identity matrix is assumed; however, this option is not preferable.
*grad	Pointer to the vg_lite_linear_gradient_t structure that contains the values to be used to fill the path.
blend	Specifies the blend mode in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Draw functions

vg_lite_draw_radial_grad function

Description:

This function is used to fill a path with a radial gradient according to the specified fill rules. The specified path is transformed according to the selected matrix and is filled with the radial color gradient. The application can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_RADIAL_GRADIENT) to determine HW support for radial gradient.

Syntax:

```
vg_lite_error_t vg_lite_draw_radial_grad (
    vg_lite_buffer_t            *target,
    vg_lite_path_t              *path,
    vg_lite_fill_t              fill_rule,
    vg_lite_matrix_t            *path_matrix,
    vg_lite_radial_gradient_t   *grad,
    vg_lite_color_t             paint_color,
    vg_lite_blend_t             blend,
    vg_lite_filter_t            filter
);
```

Parameters:

Parameter	Description
*target	Pointer to the vg_lite_buffer_t structure containing data describing the target path.
*path	Pointer to the vg_lite_path_t structure containing path data that describes the path to draw for and fill with the radial gradient. See opcode details in Vector path opcodes for plotting paths.
fill_r	Specifies the vg_lite_fill_t enum value for the fill rule for the path
*path	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If the matrix is NULL, an identity matrix is assumed; however, his option is not preferable.
*grad	Pointer to the vg_lite_radial_gradient_t structure that contains the values to be used to fill the path. **Note:** grad->image.image_mode does not support VG_LITE_MULTIPLY_IMAGE_MODE .
paint_	Specifies the paint color vg_lite_color_t RGBA value to be applied by VG_LITE_RADIAL_GRADIENT_SPREAD_FILL set by the function vg_lite_set_radial_grad. When pixels are out of the image after transformation, paint_color is applied to them. For details, see vg_lite_radial_gradient_spreadmode_t.
blend	Specifies the blend mode in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
filter	Specified the filter mode vg_lite_filter_t enum value to be applied to each drawn pixel. If no filtering is required, set this value to VG_LITE_BLEND_POINT (0).

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Draw functions

vg_lite_draw_pattern **function**

Description:

This function fills a path with an image pattern. The path is transformed according to the specified matrix and is filled with the transformed image pattern.

Syntax:

```
vg_lite_error_t vg_lite_draw_pattern (
    vg_lite_buffer_t            *target,
    vg_lite_path_t              *path,
    vg_lite_fill_t              fill_rule,
    vg_lite_matrix_t            *path_matrix,
    vg_lite_buffer_t            *pattern_image,
    vg_lite_matrix_t            *pattern_matrix,
    vg_lite_blend_t             blend,
    vg_lite_pattern_mode_t      pattern_mode,
    vg_lite_color_t             pattern_color,
    vg_lite_color_t             color,
    vg_lite_filter_t            filter
);
```

Parameters:

Parameter	Description
*target	Pointer to the vg_lite_buffer_t structure for the destination buffer. All color formats available in the vg_lite_buffer_format_t enum are valid destination formats for this draw function.
*path	Pointer to the vg_lite_path_t structure containing path data that describes the path to draw. See opcode details in Vector path opcodes for plotting paths
fill_r	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
*path	Pointer to the vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the source pixels into the target. If the matrix is NULL, an identity matrix is assumed, meaning the source is copied directly onto the target at 0,0 location.
*pattern_	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If the matrix is NULL, an identity matrix is assumed.
*pattern_	Pointer to the vg_lite_buffer_t structure that describes the source of the image pattern
	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the source pixels into the target. If the matrix is NULL, an identity matrix is assumed, which means that the source is copied directly at 0,0 location on the target.
blend	Specifies one of the vg_lite_blend_t enum values for hardware-supported blend modes to be applied to each drawn pixel in the image. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
pattern_	Specifies the vg_lite_pattern_mode_t value that defines how the region outside the image pattern is to be filled.
pattern_	Specifies a 32bpp ARGB color (vg_lite_color_t) to be applied to the fill outside the image pattern area when the pattern_mode value is VG_LITE_PATTERN_COLOR. *(from Dec 2019, type now vg_lite_color_t, previously was uint32_t)*
color	Specifies a 32bpp ARGB color (vg_lite_color_t) to be applied as a mix color. If non-zero, the mix color value gets multiplied with each source pixel before blending happens. If a mix color is not needed, set the color parameter to 0 *(from May 2023)*. **Note:** This parameter has no effect if the pattern image vg_lite_buffer_t structure member image_mode is set to VG_LITE_ZERO or VG_LITE_NORMAL_IMAGE_MODE.
filter	Specifies the filter type. All formats available in the vg_lite_filter_t enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Draw functions

Parent topic:*Vector-dased draw operations*

Linear gradient initialization and control functions

This part of the API performs linear gradient operations.

A color gradient (color progression, color ramp) is a smooth transition between a set of colors (color stops) that is done along a line (linear, or axial color gradient) or radially, along concentric circles (radial color gradient). The color transition is done by linear interpolation between two consecutive color stops.

Note: VGLite supports linear color gradients for GCNanoLiteV and GCNanoUltraV. Both linear and radial gradients are supported with GC355 and GC555.

vg_lite_init_grad **function**

Description:

This function initializes the internal buffer for the linear gradient object with default settings for rendering.

Syntax:

```
vg_lite_error_t vg_lite_init_grad (
      vg_lite_linear_gradient_t *grad
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_t structure, which defines the gradient to be initialized. Default values are used.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient initialization and control functions

vg_lite_clear_grad function

Description:

This function is used to clear the values of a linear gradient object and free up the memory of the image buffer.

Syntax:

```
vg_lite_error_t vg_lite_clear_grad (
      vg_lite_linear_gradient_t       *grad
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_t structure that is to be cleared

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient initialization and control functions

vg_lite_set_grad function

Description:

This function is used to set values for the members of the vg_lite_linear_gradient_t structure.

Note: The vg_lite_set_grad API adopts the following rules to set the default gradient colors if the input parameters are incomplete or invalid:

- If no valid stops have been specified (for example, due to an empty input array, out-of-range or out-of-order stops), a stop at 0 with (R, G, B, A) color (0.0, 0.0, 0.0, 1.0) (opaque black) and a stop at 1 with color (1.0, 1.0, 1.0, 1.0) (opaque white) are implicitly defined
- If at least one valid stop has been specified, but none has been defined with an offset of 0, then an implicit stop is added with an offset of 0 and the same color as the first user-defined stop

- If at least one valid stop has been specified, but none has been defined with an offset of 1, then an implicit stop is added with an offset of 1 and the same color as the last user-defined stop

Syntax:

```
vg_lite_error_t vg_lite_set_grad (
      vg_lite_linear_gradient_t       *grad,
      uint32_t                        count,
      uint32_t                        *colors,
      uint32_t                        *stops
     );
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_t structure to be set
count	The number of colors in the linear gradient. The maximum color stop count is defined by VLC_MAX_GRAD which is 16.
*colors	Specifies the color array for the gradient stops. The color is in ARGB8888 format with alpha in the upper byte.
*stops	Pointer to the gradient stop offset

Returns:

Always returns VG_LITE_SUCCESS.

Parent topic: Linear gradient initialization and control functions

vg_lite_get_grad_matrix function

Description:

This function is used to get a pointer to the transformation matrix of the gradient object. It allows an application to manipulate the matrix to facilitate correct rendering of the gradient path.

Syntax:

```
vg_lite_error_t vg_lite_get_grad_matrix (
      vg_lite_linear_gradient_t         *grad
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_t structure, which contains the matrix to be retrieved

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient initialization and control functions

vg_lite_update_grad function

Description:

This function is used to update or generate values for an image object that is going to be rendered. The vg_lite_linear_gradient_t object has an image buffer, which is used to render the gradient pattern. The image buffer is created or updated with the corresponding gradient parameters.

Syntax:

```
vg_lite_error_t vg_lite_update_grad (
      vg_lite_linear_gradient_t     *grad
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_t structure, which contains the update values to be used for the object to be rendered

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient initialization and control functions

Parent topic:*Vector-dased draw operations*

Linear gradient extended functions

The following functions are available only with IP that includes hardware support for extended linear gradient capabilities, such as GC355 and GC555. These functions are not available with GCNanoLiteV, GCNanoUltraV, or GCNanoV. Applications can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_LINEAR_GRADIENT_EXT) to determine HW support for linear gradient.

vg_lite_set_linear_grad function

Description:

This function is used to set the values that define the linear gradient. *(from April 2022)*

Syntax:

```
vg_lite_error_t vg_lite_set_linear_grad (
    vg_lite_ext_linear_gradient_t          *grad,
    vg_lite_uint32_t                       count,
    vg_lite_color_ramp_t                   *color_ramp,
    vg_lite_linear_gradient_parameter_t    grad_param,
    vg_lite_radial_gradient_spreadmode_t   spread_mode,
    vg_lite_uint8_t                        pre_mult
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_ext_linear_gradient_t structure that is to be set.
count	Count of the colors in the gradient. The maximum color stop count is defined by MAX_COLOR_RAMP_STOPS, which is set to 256.
*color	It is the array of stops for the linear gradient. The number of parameters for each stop is 5, and gives the offset and color of the stop. Each stop is defined by a floating-point *offset* value and four floating-point values containing the sRGBA color and alpha value associated with each stop, in the form of a non-premultiplied (R, G, B, alpha) quad. The range of all parameters is [0,1].
grad_	Gradient parameters as specified in the structure vg_lite_linear_gradient_parameter_t.
spread	The fill mode is applied to the pixels out of the paint after transformation. Uses the same spread mode enumeration types as radial gradient. For details, see vg_lite_radial_gradient_spreadmode_t enum.
pre_	This parameter controls whether color and alpha values are interpolated in premultiplied or non-premultiplied form.

Returns:

Returns VG_LITE_INVALID_ARGUMENTS to indicate the parameters are wrong.

Parent topic: Linear gradient extended functions

vg_lite_get_linear_grad_matrix **function**

Description:

This function returns a pointer to an extended linear gradient object's matrix.*(from March 2023)*.

Syntax:

```
vg_lite_matrix_t* vg_lite_get_linear_grad_matrix (
    vg_lite_ext_linear_gradient_t   *grad,
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_ext_linear_gradient_t structure.

Returns:

Returns a pointer to vg_lite_matrix_t for the specified extended linear gradient.

Parent topic: Linear gradient extended functions

vg_lite_draw_linear_grad **function**

Description:

This function returns a pointer to an extended linear gradient object's matrix.*(from March 2023)*.

Syntax:

```
vg_lite_error_t vg_lite_draw_linear_grad (
    vg_lite_buffer_t                *target,
    vg_lite_path_t                  *path,
    vg_lite_fill_t                  fill_rule,
    vg_lite_matrix_t                *path_matrix,
    vg_lite_ext_linear_gradient_t   *grad,
    vg_lite_color_t                 paint_color,
    vg_lite_blend_t                 blend,
    vg_lite_filter_t                filter
);
```

Parameters:

Parameter	Description
*target	Pointer to the vg_lite_buffer_t structure containing data describing the target path.
*path	Pointer to the vg_lite_path_t structure containing path data that describes the path to draw for the linear gradient. Refer to Vector path opcodes for plotting paths in this document for opcode detail.
fill_r	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
*path	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If the matrix is NULL, an identity matrix is assumed; however, this option is not preferable.
*grad	Pointer to the vg_lite_ext_linear_gradient_t structure that contains the values to be used to fill the path. **Note:** grad->image.image_mode does not support VG_LITE_MULTIPLY_IMAGE_MODE.
paint	Specifies the paint color vg_lite_color_t RGBA value to be applied by VG_LITE_RADIAL_GRADIENT_SPREAD_FILL, set by function vg_lite_set_linear_grad. When pixels are out of the image after transformation, this paint_color is applied to them. For details, see enum vg_lite_radial_gradient_spreadmode_t.
blend	Specifies blend mode in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
filter	Specified the filter mode vg_lite_filter_t enum value to be applied to each drawn pixel. If no filtering is required, set this value to VG_LITE_BLEND_POINT (0).

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient extended functions

vg_lite_update_linear_grad **function**

Description:

This function is used to update or generate the corresponding image object to render *(from April 2022)*.

The vg_lite_ext_linear_gradient_t object has an image buffer that is used to render the linear gradient paint. The image buffer is created/updated according to the specified grad parameters.

Syntax:

```
vg_lite_error_t vg_lite_update_linear_grad (
    vg_lite_ext_linear_gradient_t   *grad,
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_ext_t structure that is to be updated or created.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient extended functions

vg_lite_clear_linear_grad **function**

Description:

This function is used to clear the linear gradient object. This resets the grad members and free the image buffer's memory *(from April 2022)*.

Syntax:

```
vg_lite_error_t vg_lite_clear_linear_grad (
    vg_lite_ext_linear_gradient_t     *grad,
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_ext_t structure that is to be cleared.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient extended functions

Parent topic:*Vector-dased draw operations*

Radial gradient functions initialization and control functions

The following functions are available only with IP that supports radial gradients, such as GC355 and GC555. These functions are not available with GCNanoLiteV, or GCNanoUltraV or GCNanoV.

Note: There is no init function required for radial gradients. Buffer initialization is done through the vg_lite_update_radial_grad function. *(from Nov 2020, requires GC355 or GC555 hardware)*

vg_lite_set_radial_grad function

Description:

This function is used to set the values for the radial linear gradient definition. *(from November 2020, requires GC355 or GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_set_radial_grad (
    vg_lite_radial_gradient_t             *grad,
    vg_lite_uint32_t                    count,
    vg_lite_color_ramp_t                  *color_ramp,
```

(continues on next page)

(continued from previous page)

```
    vg_lite_radial_gradient_parameter_t     grad_param,
    vg_lite_radial_gradient_spreadmode_t    spread_mode,
    vg_lite_uint8_t                         pre_mult
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_radial_gradient_t structure for the radial gradient that has to be set
count	The number of color stops in the gradient. The maximum color stop count is defined by MAX_COLOR_RAMP_STOPS, which is currently 256.
*color	Pointer to the vg_lite_color_ramp_t structure that defines the stops for the radial gradient. The five parameters provide the offset and color for each stop. Each stop is defined by a set of floating point values that specify the offset and the sRGBA color and alpha values. Color channel values are in the form of a non-premultiplied (R, G, B, alpha) quad. All parameters are in the range of [0,1]. The red, green, blue, alpha value of [0, 1] is mapped to an 8-bit pixel value [0, 255].
grad_	The radial gradient parameters are supplied as a vector of 5 floats. Parameters (cx, cy) specify the center point, parameters (fx, fy) specify the focal point, and r specifies the radius. See structure vg_lite_radial_gradient_parameter_t.
spread	The tiling mode that is applied to pixels out of the paint after transformation. See enum vg_lite_radial_gradient_spreadmode_t.
pre_	Controls whether color and alpha values are interpolated in premultiplied or non-premultiplied form. If this value is set to 1, the color value of vgColorRamp is multipled by the alpha value of vgColorRamp.

Returns:

Returns VG_LITE_INVALID_ARGUMENTS to indicate that the parameters are wrong.

Parent topic: Radial gradient functions initialization and control functions

vg_lite_update_radial_grad **function**

Description:

This function is used to update or generate values for an image object that is going to be rendered. The vg_lite_radial_gradient_t object has an image buffer that is used to render the gradient pattern. The image buffer will be created or updated with the corresponding gradient parameters. *(from November 2020, requires GC355 or GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_update_radial_grad (
    vg_lite_radial_gradient_t    *grad,
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_radial_gradient_t structure, which contains the updated values to be used for the object to be rendered

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Radial gradient functions initialization and control functions

vg_lite_get_radial_grad_matrix function

Description:

This function is used to get a pointer to the radial gradient object's transformation matrix. This allows an application to manipulate the matrix to facilitate correct rendering of the gradient path*. (from Nov 2020, requires GC355 or GC555 hardware).*

Syntax:

```
vg_lite_error_t vg_lite_get_radial_grad_matrix (
    vg_lite_radial_gradient_t     *grad,
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_radial_gradient_t structure, which contains the matrix to be retrieved

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Radial gradient functions initialization and control functions

vg_lite_clear_rad_grad function

Description:

This function is used to clear the values of a radial gradient object and free the image buffer's memory*. (from Nov 2020, requires GC355 or GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_clear_radial_grad (
    vg_lite_radial_gradient_t    *grad,
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_radial_gradient_t structure which is to be cleared

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Radial gradient functions initialization and control functions

Parent topic:*Vector-dased draw operations*

Stroke operations This part of the API performs stroke operations. *(from March 2022)*

Stroke enumerations

This section gives details on stroke enumerations.

vg_lite_cap_style_t **enumeration**

Defines the style of cap at the end of a stroke *(from March 2022)*.

Used in structure: vg_lite_stroke_t.

Used in function: vg_lite_set_stroke.

vg_lite_cap values	Description
VG_LITE_	The *butt* end cap style terminates each segment with a line perpendicular to the tangent at each endpoint.
VG_LITE_	The *round* end cap style appends a semicircle with a diameter equal to the line width centered around each endpoint.
VG_LITE_	The *square* end cap style appends a rectangle with two sides of length equal to the line width perpendicular to the tangent, and two sides of length equal to half the line width parallel to the tangent, at each endpoint.

Parent topic: Stroke enumerations

vg_lite_path_type_t **enumeration**

Defines the type of draw path *(from March 2022)*.

Used in structure: vg_lite_path_t, vg_lite_stroke_t.

Used in function: vg_lite_set_path_type.

vg_lite_path_type_t string values	Description
VG_LITE_DRAW_FILL_PATH	Draw path is fill.
VG_LITE_DRAW_STROKE_PATH	Draw path is stroke.
VG_LITE_DRAW_FILL_STROKE_PATH	Draw path is both fill and stroke.

Parent topic: Stroke enumerations

vg_lite_join_style_t **enumeration**

Defines the type of styles available for line joints. *(from March 2022)*

Used in structure: vg_lite_stroke_t.

Used in function: vg_lite_set_stroke.

vg_lite_joi string values	Description
VG_LITE_	The *miter* join style appends a trapezoid with one vertex at the intersection point of the two original lines, two adjacent vertices at the outer endpoints of the two “thickened” lines and a fourth vertex at the extrapolated intersection point of the outer perimeters of the two “thickened” lines.
VG_LITE_	The *round* join style appends a wedge-shaped portion of a circle, centered at the intersection point of the two original lines, having a radius equal to half the line width.
VG_LITE_	The *bevel* type join style appends a triangle with two vertices at the outer endpoints of the two “thickened” lines and a third vertex at the intersection point of the two original lines.

Parent topic: Stroke enumerations

Parent topic:*Stroke operations*

Stroke structures

This section gives details on stroke structures.

vg_lite_path_t structure

Defined under Vector Path Structures - vg_lite_path_t structure.

(additional members added for stroke from March 2022)

Parent topic: Stroke structures

vg_lite_path_list_t structure

The structure vg_lite_path_list_ptr points to the vg_lite_path_list structure that provides divided path data according to MOVE/MOVE_REL. *(from Aug 2023)*

Used (vg_lite_path_list_ptr) in structures: vg_lite_stroke_t.

vg_lite_path_list_t members	Type	Description
path_points	vg_lite_path_point_ptr	
path_end	vg_lite_path_point_ptr	
point_count	vg_lite_uint32_t	
next	vg_lite_path_list_ptr	
closed	vg_lite_uint8_t	

Parent topic: Stroke structures

vg_lite_path_point_t structure

The structure vg_lite_path_point_ptr points to the vg_lite_path_point structure which provides path detail *(from March 2022)*

Used (vg_lite_path_point_ptr) in structures: vg_lite_path_point_t, vg_lite_stroke_conversion. vg_lite_sub_path_t.

vg_lite_path_point_t members	Type	Description
x	vg_lite_float_t	X coordinate
y	vg_lite_float_t	Y coordinate
flatten_flag	vg_lite_uint8_t	Flatten flag for flattened path
curve_type	vg_lite_uint8_t	Curve type for the stroke path
tangentX	vg_lite_float_t	X tangent (Note: #define centerX tangent)
tangentY	vg_lite_float_t	Y tangent (Note: #define centerX tangent)
length	vg_lite_float_t	Line length
prev	vg_lite_path_point_pt	Pointer to the previous point node

Parent topic: Stroke structures

vg_lite_stroke_t structure

The structure provides stroke parameters and pointers to temp storage for a stroke sub path. Refer to the function vg_lite_set_stroke parameter descriptions for additional description for some members. *(from March 2022)*

Used in structure: vg_lite_path_t.

vg_lite_stroke_t members	Type	Description
cap_style	vg_lite_cap_style_t	Stroke cap style
join_style	vg_lite_join_style_t	Stroke joint style
line_width	vg_lite_float_t	Stroke line width
miter_limit	vg_lite_float_t	Stroke miter limit
*dash_pattern	vg_lite_float_t	Pointer to stroke dash pattern
pattern_count	vg_lite_uint32_t	Number of dash pattern repetitions
dash_phase	vg_lite_float_t	Stroke dash phrase
dash_length	vg_lite_float_t	Stroke dash initial length
dash_index	vg_lite_uint32_t	Stroke dash initial index
half_width	vg_lite_float_t	Half line width
pattern_length	vg_lite_float_t	Total length of stroke dash patterns.
miter_square	vg_lite_float_t	For fast checking
path_points	vg_lite_path_point_ptr	Temp storage for stroke sub path
path_end	vg_lite_path_point_ptr	Temp storage for stroke sub path
point_count	unint32_t	Temp storage for stroke sub path
left_point	vg_lite_path_point_ptr	Temp storage for stroke sub path
right_pont	vg_lite_path_point_ptr	Temp storage for stroke sub path
stroke_points	vg_lite_path_point_ptr	Temp storage for stroke sub path
stroke_end	vg_lite_path_point_ptr	Temp storage for stroke sub path
stroke_count	vg_lite_uint32_t	Temp storage for stroke sub path
path_list_divide	vg_lite_path_list_ptr	Divide stroke path according to move or move_rel for
cur_list	vg_lite_path_list_ptr	Pointer to current divided path data. *(from Aug 2023)*
add_end	vg_lite_uint8_t	Flag that adds end_path in driver *(from Aug 2023)*
dash_reset	vg_lite_uint8_t	*(from Aug 2023)*
stroke_paths	vg_lite_sub_path_ptr	
last_stroke	vg_lite_sub_path_ptr	
swing_handling	vg_lite_uint32_t	
swing_deltax	vg_lite_float_t	
swing_deltay	vg_lite_float_t	
swing_start	vg_lite_path_point_ptr	
swing_stroke	vg_lite_path_point_ptr	

Table 51 – continued from previous page

vg_lite_stroke_t members	Type	Description
swing_length	vg_lite_float_t	
swing_centlen	vg_lite_float_t	
swing_count	vg_lite_uint32_t	
need_swing	vg_lite_uint8_t	
swing_ccw	vg_lite_uint8_t	
stroke_length	vg_lite_float_t	
stroke_size	vg_lite_uint32_t	
fattened	vg_lite_uint8_t	The stroke line is a fat line.
closed	vg_lite_uint8_t	

Parent topic: Stroke structures

vg_lite_sub_path_t **structure**

The structure vg_lite_sub_path_ptr points to the vg_lite_sub_path structure that provides sub path detail and a pointer to the next sub path. *(from March 2022)*

Used in structure: vg_lite_stroke_conversion.

vg_lite_path_point_t members	Type	Description
next	vg_lite_sub_path_ptr	Pointer to the next sub path
point_count	vg_lite_uint32_t	Number of points in the sub path
point_list	vg_lite_path_point_pt	Pointer to the point list.
end_point	vg_lite_path_point_pt	Pointer to the last point.
closed	vg_lite_uint8_t	Indicates whether or not the path is closed.
length	vg_lite_float_t	Length of the sub path.

Parent topic: Stroke structures

Parent topic:*Stroke operations*

Stroke functions

All return vg_lite_error_t status.

vg_lite_set_path_type **function**

Description:

This function sets the path type*. (from March 2022)*

Syntax:

```
vg_lite_error_t vg_lite_set_path_type (
    vg_lite_path_t              *path,
    vg_lite_path_type_t         path_type
);
```

Parameters:

Parameter	Description
*path	Pointer to the **vg_lite_path_t** structure that describes the vector path.
path_type	Pointer to a vg_lite_path_type_t structure that describes the path type.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Stroke functions

vg_lite_set_stroke **function**

Description:

This function uses input parameters to set stroke attributes *(from March 2022)*.

Syntax:

```
vg_lite_error_t vg_lite_set_stroke (
    vg_lite_path_t              *path,
    vg_lite_cap_style_t          cap_style,
    vg_lite_join_style_t         join_style,
    vg_lite_float_t             line_width,
    vg_lite_float_t             miter_limit,
    vg_lite_float_t             *dash_pattern,
    vg_lite_uint32_t             pattern_count,
    vg_lite_float_t             dash_phase,
    vg_lite_color_t              color
);
```

Parameters:

Parameter	Description
*path	Pointer to the vg_lite_path_t structure that describes the path.
cap_style	The end cap style is defined by the vg_lite_cap_style_t enum.
join_style	The line join style defined by the vg_lite_join_style_t enum.
line_width	The line width of the stroke path. A line width less than or equal to 0 prevents stroking from taking place.
miter_limit	When stroking using the Miter stroke vg_lite_join_style_t, the miter length (that is, the length between the intersection points of the inner and outer perimeters of the two “fattened” lines) is compared to the product of the user-set miter limit and the line width. If the miter length exceeds this product, the Miter join is not drawn and a Bevel join is substituted. **Note**: Miter limit values less than 1 are silently clamped to 1.
*dash_pattern	Pointer to a dash pattern that consists of a sequence of lengths of alternating “on” and “off” dash segments. The first value of the dash array defines the length, in user coordinates, of the first “on” dash segment. The second value defines the length of the following “off” segment. Each subsequent pair of values defines one “on” and one “off” segment. **Note**: If the dash pattern has an odd number of elements, the final element is ignored.
pattern_count	The count of dash on/off segments.
dash_phase	Defines the starting point in the dash pattern that is associated with the start of the first segment of the path. For example, if the dash pattern is [10 20 30 40] and the dash phase is 35, the path is stroked with an “on” segment of length 25 (skipping the first “on” segment of length 10, the following “off” segment of length 20, and the first 5 units of the next “on” segment), followed by an “off” segment of length 40. The pattern is then repeated from the beginning, with an “on” segment of length 10, an “off” segment of length 20, an “on” segment of length 30.
color	The stroke color.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Stroke functions

vg_lite_update_stroke **function**

Description:

This function uses the path and stroke attributes as specified with the function vg_lite_set_stroke to update the stroke path's parameters and generate stroke path data . *(from March 2022)*

Syntax:

```
vg_lite_error_t vg_lite_update_stroke (
    vg_lite_path_t              *path,
);
```

Parameters:

Parameter	Description
*path	Pointer to the **vg_lite_path_t** structure that describes the path.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Stroke functions

Parent topic:*Stroke operations*

Deprecated and renamed APIs The following functions are deprecated and are either obsolete or replaced by a more efficient implementation. Their use is discouraged and will produce unpredictable behaviors.

The names of some functions, enums and structures were modified during code refinements in 2022Q3. If the parameters did not change, the deprecated syntax detail is not provided below. Changes to enums and structs are not mentioned here, instead refer to the item itself.

Deprecated or renamed API	Recommended replacement API	Source file	Date deprecated
vg_lite_perspective	n/a	vg_lite.h	August 2022
vg_lite_set_dither	vg_lite_enable_dither vg_lite_disable_dither	vg_lite.h	August 2022
vg_lite_append_path	vg_lite_path_append	vg_lite.h	Sept 2022
vg_lite_path_calc_length	vg_lite_get_path_length	vg_lite.h	Sept 2022
vg_lite_set_image_global_alpha	vg_lite_set_source_global_alpha	vg_lite.h	Sept 2022
vg_lite_dest_global_alpha	vg_lite_set_dest_global_alpha	vg_lite.h	Sept 2022
vg_lite_mem_avail	vg_lite_get_mem_size	vg_lite.h	Sept 2022
vg_lite_enable_premultiply	n/a	vg_lite.h	Dec 2022
vg_lite_disable_premultiply	n/a	vg_lite.h	Dec 2022
vg_lite_set_premultiply	n/a	vg_lite.h	Aug 2023
vg_lite_radial_gradient_spreadmo enum	vg_lite_gradient_spreadmode_t enum	vg_lite.h	March 2023
API Name Refinement	***(no change to parameters)***		
vg_lite_buffer_upload	vg_lite_upload_buffer_	vg_lite.h	Sept 2022
vg_lite_*mask*	*most vg_lite_*mask_layer*	vg_lite.h	Sept 2022
vg_lite_*_grad	vg_lite_*_gradient *(parameters unchanged)*	vg_lite.h	Sept 2022
vg_lite_*_radial_grad*	vg_lite_*_rad_grad*	vg_lite.h	Sept 2022
vg_lite_buffer_image_mode_t	vg_lite_image_mode_t	vg_lite.h	Sept 2022
vg_lite_transparency_mode_t	vg_lite_transparency_t	vg_lite.h	Sept 2022
vg_lite_set_update_stroke	vg_lite_update_stroke	vg_lite.h	Sept 2022
vg_lite_set_draw_path_type	vg_lite_set_path_type	vg_lite.h	Sept 2022

Deprecated vg_lite syntax

Syntax for deprecated functions is provided below for reference.

Note: This list does not include items renamed during code refinement of Sept 2022.

vg_lite_perspective ***(deprecated)***

Syntax:

```
void vg_lite_perspective (
    vg_lite_float_t             px,
    vg_lite_float_t             py,
    vg_lite_matrix_t            *matrix
);
```

Parent topic: Deprecated vg_lite syntax

vg_lite_set_dither ***(deprecated)***

Syntax:

```
vg_lite_error_t vg_lite_set_dither (
    int                 enable
);
```

Parent topic: Deprecated vg_lite syntax

vg_lite_enable_premultiply ***(deprecated)***

Syntax:

```
vg_lite_error_t vg_lite_enable_premultiply (
    void
);
```

Parent topic: Deprecated vg_lite syntax

vg_lite_disable_premultiply ***(deprecated)***

Syntax:

```
vg_lite_error_t vg_lite_disable_premultiply (
    void
);
```

Parent topic: Deprecated vg_lite syntax

vg_lite_set_premultiply ***(deprecated)***

Syntax:

```
vg_lite_error_t vg_lite_set_premultiply (
    vg_lite_uint8_t             src_premult,
    vg_lite_uint8_t             dst_premult,
);
```

Parent topic: Deprecated vg_lite syntax

Parent topic:*Deprecated and renamed APIs*

VGLite API version 2.0 to 3.0 migration guide The VGLite API version 3.0 is not fully compatible with VGLite API version 2.0. VGLite API version 3.0 includes some new API functions for the new features in the latest VG GPU like GC555. Some VGLite API version 2.0 function interfaces are changed in API version 3.0. So, the existing VGLite API version 2.0 applications must be modified to compile and run properly with the VGLite API version 3.0 driver. This chapter provides guidance for migrating VGLite API version 2.0 applications to VGLite API version 3.0.

VGLite API name changes in API version 3.0

Some original VGLite API names are changed in API version 3.0 for API naming consistency. In the VGLite API version 3.0 header file vg_lite.h, a set of API name macros are defined for the equivalent API names between API version 3.0 and API version 2.0, so it is not necessary to modify the VGLite API function names in API version 2.0 applications for the application to compile and run with the API version 3.0 driver.

The list of equivalent VGLite API functions between API version 3.0 and API version 2.0 is shown below. These API functions' parameters are the same between API version 3.0 and API version 2.0.

```
/* API name defines for backward compatibility to VGLite 2.0 APIs */
#define vg_lite_buffer_upload                   vg_lite_upload_buffer
#define vg_lite_path_append                     vg_lite_append_path
#define vg_lite_path_calc_length                vg_lite_get_path_length
#define vg_lite_set_ts_buffer                   vg_lite_set_tess_buffer
#define vg_lite_set_draw_path_type              vg_lite_set_path_type
#define vg_lite_create_mask_layer               vg_lite_create_masklayer
#define vg_lite_fill_mask_layer                 vg_lite_fill_masklayer
#define vg_lite_blend_mask_layer                vg_lite_blend_masklayer
#define vg_lite_generate_mask_layer_by_path     vg_lite_render_masklayer
#define vg_lite_set_mask_layer                  vg_lite_set_masklayer
#define vg_lite_destroy_mask_layer              vg_lite_destroy_masklayer
```

(continues on next page)

(continued from previous page)

```
#define vg_lite_enable_mask                  vg_lite_enable_masklayer
#define vg_lite_enable_color_transformation     vg_lite_enable_color_transform
#define vg_lite_set_color_transformation       vg_lite_set_color_transform
#define vg_lite_set_image_global_alpha          vg_lite_source_global_alpha
#define vg_lite_set_dest_global_alpha          vg_lite_dest_global_alpha
#define vg_lite_clear_rad_grad                vg_lite_clear_radial_grad
#define vg_lite_update_rad_grad                vg_lite_update_radial_grad
#define vg_lite_get_rad_grad_matrix            vg_lite_get_radial_grad_matrix
#define vg_lite_set_rad_grad                  vg_lite_set_radial_grad
#define vg_lite_draw_linear_gradient           vg_lite_draw_linear_grad
#define vg_lite_draw_radial_gradient           vg_lite_draw_radial_grad
#define vg_lite_draw_gradient                 vg_lite_draw_grad
#define vg_lite_mem_avail                    vg_lite_get_mem_size
#define vg_lite_set_update_stroke              vg_lite_update_stroke
```

The list of equivalent VGLite API structures and enumerations is shown below:

```
#define vg_lite_buffer_image_mode_t          vg_lite_image_mode_t
#define vg_lite_draw_path_type_t             vg_lite_path_type_t
#define vg_lite_linear_gradient_ext_t        vg_lite_ext_linear_gradient_t
#define vg_lite_buffer_transparency_mode_t     vg_lite_transparency_t
```

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

vg_lite_set_scissor API interface change

The VGLite API vg_lite_set_scissor() function name is not changed in API version 3.0, but the API parameters are defined differently in API version 3.0.

In VGLite API version 3.0, the vg_lite_set_scissor() function is defined as:

```
/* Set and enable a scissor rectangle for render target. */
vg_lite_error_t vg_lite_set_scissor(vg_lite_int32_t x, vg_lite_int32_t y,
            vg_lite_int32_t right, vg_lite_int32_t bottom);
```

In VGLite API version 2.0, the vg_lite_set_scissor() function is defined as:

```
vg_lite_error_t vg_lite_set_scissor(int32_t x, int32_t y, int32_t width, int32_t height);
```

So, the vg_lite_set_scissor() API parameters “width” and “height” in the VGLite API version 2.0 application must be changed to “right” x-coordinate value and “bottom” y-coordinate value.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

vg_lite_map API interface change

The VGLite API vg_lite_map() function name is not changed in API version 3.0, but the API parameters are defined differently in API version 3.0.

In VGLite API version 3.0, the vg_lite_map() function is defined as:

```
/* Map a buffer into hardware accessible address space. */
vg_lite_error_t vg_lite_map(vg_lite_buffer_t *buffer, vg_lite_map_flag_t flag, int32_t fd);
```

In VGLite API version 2.0, the vg_lite_map() function is defined as:

```
vg_lite_error_t vg_lite_map(vg_lite_buffer_t *buffer);
```

So, vg_lite_map() in VGLite API version 3.0 API requires two extra parameters “flag” and “fd”, which can simply be set as vg_lite_map (buffer, 0, 0) in applications.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

vg_lite_enable_scissor / vg_lite_disable_scissor API

The VGLite API vg_lite_enable_scissor() and vg_lite_disable_scissor() functions are valid only for vg_lite_scissor_rects() API. They have no effect for vg_lite_set_scissor() in VGLite API version 3.0.

Although the behavior of vg_lite_enable_scissor() and vg_lite_disable_scissor() is changed in VGLite API version 3.0, there is no need to change these functions in VGLite API version 2.0 applications to work with the VGLite API version 3.0 driver.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

vg_lite_draw_pattern API interface change

The VGLite API vg_lite_draw_pattern() function name is not changed in API version 3.0, but the API parameters are defined differently in API version 3.0.

In VGLite API version 3.0, the vg_lite_draw_pattern() function is defined as:

```
/* Draw a path that is filled by a transformed image pattern. */
vg_lite_error_t vg_lite_draw_pattern(vg_lite_buffer_t *target,
                                     vg_lite_path_t *path,
                                     vg_lite_fill_t fill_rule,
                                     vg_lite_matrix_t *path_matrix,
                                     vg_lite_buffer_t *pattern_image,
                                     vg_lite_matrix_t *pattern_matrix,
                                     vg_lite_blend_t blend,
                                     vg_lite_pattern_mode_t pattern_mode,
                                     vg_lite_color_t  pattern_color,
                                     vg_lite_color_t  color,
                                     vg_lite_filter_t filter);
```

Compared to the VGLite API version 2.0 vg_lite_draw_pattern() function, “color” is a new additional parameter. It specifies a 32bpp ARGB color (vg_lite_color_t) to be applied as a mix color. If nonzero, the mix color value gets multiplied with each source pixel before blending happens. If a mix color is not needed, set the color parameter to 0.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

[New] vg_lite_copy_image in VGLite API version 3.0

The new API vg_lite_copy_image() is added in VGLite API version 3.0 to support the OpenVG vgCopyImage API, which performs a pixel rectangle copy without pixel transformation, blending, filtering operations.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

vg_lite_set_dither API is deprecated in API version 3.0

The original API version 2.0 function vg_lite_set_dither(int enable) API is removed from API version 3.0, it is replaced with two new APIs for dither enable/disable:

```
/* Enable dither function. Dither is OFF by default. */
vg_lite_error_t vg_lite_enable_dither();
/* Disable dither function. Dither is OFF by default. */
vg_lite_error_t vg_lite_disable_dither();
```

Therefore, the vg_lite_set_dither(enable) function in the VGLite API version 2.0 application must be replaced with vg_lite_enable_dither() or vg_lite_disable_dither() to work with the VGLite API version 3.0 driver.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

Deprecated VGLite API version 2.0 functions

The VGLite API vg_lite_perspective(), vg_lite_enable_premultiply(), vg_lite_disable_premultiply() functions are removed from API version 3.0. These API functions must be deleted from a VGLite API version 2.0 application to work with the VGLite API version 3.0 driver.

In VGLite API version 3.0, the color premultiply setting is defined by the vg_lite_blend_t enumeration to replace the original vg_lite_enable_premultiply() and vg_lite_disable_premultiply() APIs.

- VG_LITE_BLEND_* enumeration values in vg_lite_blend_t define non-premultiplied blending modes.
- OPEVG_BLEND_* enumeration values in vg_lite_blend_t define premultiplied Porter-Duff blending modes.

So, the VGLite API version 3.0 application can set different blending modes to get the desired premultiplied/non-premultiplied blending result.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

Revision history

Document ID	Release date	Description
IMX Rev. 1.2	17 January 2025	The document is updated to correspond to the API version 3.0
IMX Rev. 1.1	22 September 2022	- Paragraph 4.1.1 Updated *Table 3* - *vg_lite_feature_t* enumeration. - Paragraph 6.6 Added documentation for new API *vg_lite_set_dither* - Paragraph 8.2 Blit structures- Added documentation for new data structure *vg_lite_color_key_t* -; added documentation for new data structure *vg_lite_color_key4_t* - Paragraph 8.3.1, *vg_lite_blit* function- added note related to HW limitation on RT500 platform - Paragraph 8.3.2, *vg_lite_blit_rect* function -added note related to HW limitation on RT500 platforms - Paragraph 8.3.3, *vg_lite_get_transform_matrix* function- adjusted function description, adjusted function parameters description - Paragraph 8.3, blit functions- added documentation for new API *vg_lite_set_color_key* - Paragraph 8.4.1, *vg_lite_enable_premultiply* function- added note about limited support on specific platforms - Paragraph 8.4.2, *vg_lite_disable_premultiply* function- added note about limited support on specific platforms - Paragraph 10.1.3, *vg_lite_fill_t* enumeration- added note about crossing points buffer limitation - Paragraph 10.2, draw and gradient structures- added documentation for new data structure *vg_lite_gradient_parameter_t - done*- added documentation for new data structure *vg_lite_gradient_ext_t*- Paragraph 10.3, draw functions- added documentation for new API *vg_lite_draw_linear_gradient*- Paragraph 10, vector-Based Draw Operations - added new paragraph 10.5 *Extended linear gradient initialization and control functions*; added documentation for new API *vg_lite_set_linear_gradient*; added documentation for new API *vg_lite_get_linear_grad_matrix*; added documentation for new API *vg_lite_update_linear_grad*; Added documentation for new API *vg_lite_clear_linear_grad* - Paragraph 10.5, Radial gradient functions - adjusted paragraph title - Added new Chapter *Stroke Operations* - Chapter *Platform-Specific Features* -updated *Table 41* - Platform-specific VGLite features
IMX Rev. 1	27 January 2022	*Introduction* Added i.MX RT1160 to the list of NXP devices that support VGLite graphics API vg_lite_error_t enumeration Updated Table 1 vg_lite_feature_t enumerationUpdated Table 1 *API control*
IMX Rev. 0	22 February 2021	Initial release

Note about the source code in the document Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

3.7.2 Xtensa Audio Framework (XAF)

Xtensa Audio Framework (XAF) Examples

Overview The Xtensa Audio Framework (XAF) is designed to accelerate the development of audio processing applications for the HiFi family of DSP cores. The multicore version of XAF described in these examples is designed to work with subsystems having single or multiple DSPs, enabling sophisticated audio processing capabilities in embedded systems.

Each demo showcases a dual-core architecture:

- cm33/ - The ARM application for the Cortex-M33 core, which provides the user interface and system control
- dsp/ - The DSP application that performs audio processing using the XAF middleware library

When an application is started, a shell interface is displayed on the terminal that executes from the ARM core. Users can control the application through shell commands, which are relayed via RPMsg-Lite IPC to the DSP core where they are processed and responses are returned. This architecture demonstrates efficient partitioning of workloads - with user interface and control tasks handled by the ARM core while computationally intensive audio processing is offloaded to the specialized DSP core.

For more information about XAF and detailed documentation on the API and available components, please refer to the Cadence XAF documentation (/middleware/cadence/multicore-xaf/xa_af_hostless/doc).

Availability Note **Important**: These XAF examples are not included in the standard MCUXpresso SDK repository. They are available as part of the MCUXpresso SDK Builder package on the NXP website. To access these examples, please visit MCUXpresso SDK Builder and create a customized SDK package that includes the XAF examples for your target platform.

Included Examples XAF Playback Example

The dsp_xaf_playback application demonstrates audio file decoding and playback capabilities using the DSP core and Xtensa Audio Framework, supporting various audio codecs while handling operations through a shell interface on the ARM core that communicates with DSP processing.

XAF Record Example

The dsp_xaf_record example captures audio from digital microphones (DMIC), processes it on the DSP core using voice enhancement algorithms, performs voice recognition (VIT), and outputs the detected wake words and voice commands to the console, enabling hands-free voice control applications.

XAF USB Example

The XAF USB example demonstrates DSP-powered USB audio processing in two configurations: USB speaker and USB microphone. The application uses shell commands to switch between modes, with the ARM core handling USB communication while the DSP processes audio.

XAF Playback Example

Table of Content

- *Overview*
- *Functionality*
- *Hardware Requirements*
- *Hardware Modifications*
- *Preparation*
- *Example Configuration*
- *Running the Demo*
- *Known Issues*

Overview The dsp_xaf_playback application demonstrates audio processing using the DSP core, the Xtensa Audio Framework (XAF) middleware library, and selected Xtensa audio codecs.

As shown in the table below, the application is supported on several development boards and each development board may have certain limitations, some development boards may also require hardware modifications or allow the use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- **MP3 encoder, G.711, G.722, BSAC, DAB+, DAB/MP2, DRM:** Provided only as linked libraries but are not enabled in the example.

Functionality The application includes the following main components:

1. **ARM Core (CM33)** - Handles user interface, SD card operations, and communicates with the DSP core
2. **DSP Core** - Processes audio data using the Xtensa Audio Framework (XAF)

The typical audio processing pipeline includes:

- File source component (reads from SD card)
- Decoder component (decodes compressed audio)
- Renderer component (outputs to audio hardware)

When the file playback command is issued, the ARM core reads the file from SD card and sends data to the DSP, which processes it and outputs to the audio hardware.

Hardware Requirements

- Development board (one of the following):
  - EVK-MIMXRT595 board
  - EVK-MIMXRT685 board

  - MIMXRT685-AUD-EVK board
  - MIMXRT700-EVK board
- Micro USB cable
- JTAG/SWD debugger
- Headphones with 3.5 mm stereo jack
- Personal Computer
- SD card with audio files (for file playback feature)

Hardware Modifications Some development boards need some hardware modifications to run the application.

- *EVK-MIMXRT595:*

  To enable the example audio using WM8904 codec, connect pins as follows:

  - JP7-1 <–> JP8-2

  Note: The I3C Pin configuration in pin_mux.c is verified for default 1.8V, for 3.3V, need to manually configure slew rate to slow mode for I3C-SCL/SDA.

- *EVK-MIMXRT685:*

  To enable the example audio using WM8904 codec, connect pins as follows:

  - JP7-1 <–> JP8-2

- *MIMXRT685-AUD-EVK:*
  - Set the hardware jumpers (Tower system/base module) to default settings.
  - Set hardware jumpers JP2 2<–>3, JP44 1<–>2 and JP45 1<–>2.
- *MIMXRT700-EVK:*

  Set the hardware jumpers to default settings.

Preparation

1. Connect headphones to Audio HP / Line-Out connector (J4).
   - EVK-MIMXRT595 - J4
   - EVK-MIMXRT685 - J4
   - MIMXRT685-AUD-EVK - J4, J50, J51, J52
   - MIMXRT700-EVK - J29
2. Connect a micro USB cable between the PC host and the debug USB port on the development board.
   - EVK-MIMXRT595 - J40
   - EVK-MIMXRT685 - J5
   - MIMXRT685-AUD-EVK - J5
   - MIMXRT700-EVK - J54
3. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity

- One stop bit
- No flow control

4. Download the program for CM33 core to the target board.
5. Launch the debugger in your IDE to begin running the demo.
6. If building release configuration, start the xt-ocd daemon and download the program for DSP core to the target board. If building debug configuration, launch the Xtensa IDE or xt-gdb debugger to begin running the demo.

Notes:

- DSP image can only be debugged using J-Link debugger. See the document 'Getting Started with Xplorer' for your particular board for more information.

Example Configuration The example can be configured by user. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **MIMXRT700-EVK Decoder Configuration:**

  RT700 has limited RAM on Cortex-M33 core 1 which limits the available decoders. Only SBC decoder is enabled by default. In order to enable a different decoder/encoder, it is necessary to define the appropriate define on project level. Use one of the following define from the list of the supported decoders on the HiFi1 core:

  – XA_AAC_DECODER
  – XA_MP3_DECODER
  – XA_SBC_DECODER
  – XA_VORBIS_DECODER
  – XA_OPUS_DECODER

Running the Demo The ARM application will power and clock the DSP, so it must be loaded prior to loading the DSP application. The DSP application can be built by the following tools: Xtensa Xplorer or Xtensa C Compiler. Application for Cortex-M33 can be built by the other toolchains listed in MCUXpresso SDK Release Notes.

The release configurations of the demo will combine both applications into one ARM image. With this, the ARM core will load and start the DSP application on startup. Pre-compiled DSP binary images are provided under dsp/binary/ directory. If you make changes to the DSP application in release configuration, rebuild ARM application after building the DSP application. If you plan to use MCUXpresso IDE for cm33 you will have to make sure that the preprocessor symbol DSP_IMAGE_COPY_TO_RAM, found in IDE project settings, is defined to the value 1 when building release configuration.

The debug configurations will build two separate applications that need to be loaded independently. DSP application can be built by the following tools: Xtensa Xplorer or Xtensa C Compiler. Required tool versions can be found in MCUXpresso SDK Release Notes for the board. Application for cm33 can be built by the other toolchains listed there. If you plan to use MCUXpresso IDE for cm33 you will have to make sure that the preprocessor symbol DSP_IMAGE_COPY_TO_RAM, found in IDE project settings, is defined to the value 0 when building debug configuration. The ARM application will power and clock the DSP, so it must be loaded prior to loading the DSP application.

In order to debug both the Cortex-M33 and DSP side of the application, please follow the instructions:

1. It is necessary to run the Cortex-M33 side first and stop the application before the DSP_Start function

2. Run the xt-ocd daemon with proper settings
3. Download and debug the DSP application

In order to get TRACE debug output from the XAF it is necessary to define XF_TRACE 1 in the project settings. It is possible to save the TRACE output into RAM using DUMP_TRACE_TO_BUF 1 define on project level. Please see the initialization of the TRACE function in the xaf_main_dsp.c file. For more details see XAF documentation.

When the demo runs successfully, the terminal will display the following output (example from MIMXRT700-EVK):

```
******************************
DSP audio framework demo start
******************************

[CM33 Main] Configure codec

[DSP_Main] Cadence Xtensa Audio Framework
[DSP_Main] Library Name    : Audio Framework (Hostless)
[DSP_Main] Library Version : 3.5
[DSP_Main] API Version     : 3.2

[DSP_Main] start
[DSP_Main] established RPMsg link
[CM33_Main] DSP image copied to DSP TCM
[CM33_Main][APP_SDCARD_Task] start
[CM33_Main][APP_DSP_IPC_Task] start
[CM33_Main][APP_Shell_Task] start

Copyright  2024  NXP
```

Type help to see the command list. Similar description will be displayed on serial console (*If multi-channel playback mode is enabled, the description is slightly different. Available encoders/decoders may differ - refer to the* table.):

```
"help": List all the registered commands

"exit": Exit program

"version": Query DSP for component versions

"file": Perform audio file decode and playback from SD card
  USAGE: file [list|stop|<audio_file>]
    list          List audio files on SD card available for playback
    <audio_file>  Select file from SD card and start playback

"decoder": Perform decode on DSP and play to speaker.
  USAGE: decoder [aac|mp3|opus|sbc|vorbis_ogg|vorbis_raw]
    aac:        Decode aac data
    mp3:        Decode mp3 data
    opus:       Decode opus data
    sbc:        Decode sbc data
    vorbis_ogg: Decode OGG VORBIS data
    vorbis_raw: Decode raw VORBIS data

"encoder": Encode PCM data on DSP and compare with reference data.
  USAGE: encoder [opus|sbc]
    opus: Encode pcm data using opus encoder
    sbc:  Encode pcm data using sbc encoder

"src" Perform sample rate conversion on DSP
```

(continues on next page)

(continued from previous page)

```
"gain": Perform PCM gain adjustment on DSP
```

Xtensa IDE log when command is playing a file (mp3/aac/vorbis/...):

```
  File playback start, initial buffer size: 16384
  [DSP Codec] Audio Device Ready
  [DSP Codec] Decoder component started
  [DSP Codec] Setting decode playback format:
    Decoder    : mp3_dec
    Sample rate: 16000
    Bit Width  : 16
    Channels   : 2
  [DSP Codec] Renderer component started
  [DSP Codec] Connected XA_DECODER ->  XA_RENDERER
  [DSP_ProcessThread] start
  [DSP_BufferThread] start
```

Xtensa IDE log when decoder command starts playback successfully:

```
  [DSP_Main] Input buffer addr: 0x20020000, buffer size: 94276
  [DSP Codec] Audio Device Ready
  [DSP Codec] Decoder created
  [DSP Codec] Decoder component started
  [DSP Codec] Renderer component created
  [DSP Codec] Connected XA_DECODER -> XA_RENDERER
  [DSP_ProcessThread] start
  [DSP_ProcessThread] Execution complete - exiting
  [DSP_ProcessThread] exiting
  [DSP Codec] Audio device closed

  [CM33 CMD] [APP_DSP_IPC_Task] response from DSP, cmd: 0, error: 0
  [CM33 CMD] Decode complete
```

MIMXRT685-AUD-EVK Multi-channel Support:

The MIMXRT685-AUD-EVK board supports multi-channel audio. When selecting audio files for playback, you can specify the number of channels:

```
```
file [list|stop|<audio_file> [<nchannel>]]
<nchannel>   Select the number of channels (2 or 8 can be selected).
NOTE: Selected audio file must meet the following parameters:
          - Sample rate: 96 kHz
          - Width: 32 bit
```
```

Xtensa IDE log when command is playing a PCM file:

```
```
[DSP_FILE_REN] Audio Device Ready
[DSP_FILE_REN] post-proc/pcm_gain component started
[DSP_FILE_REN] post-proc/client_proxy component started
[DSP_FILE_REN] Connected post-proc/pcm_gain -> post-proc/client_proxy
[DSP_FILE_REN] renderer component started
[DSP_FILE_REN] Connected post-proc/client_proxy -> renderer
[DSP_BufferThread] start
[DSP_ProcessThread] start
[DSP_CleanupThread] start3
```
```

Known Issues

1. The “file stop” command doesn’t stop the playback for some small files (with low sample rate).
2. MIMXRT700-EVK: Has limited RAM on Cortex-M33 core 1 which limits the available decoders.

XAF Record Example

Table of Content

- *Overview*
- *Functionality*
- *Hardware Requirements*
- *Hardware Modifications*
- *Preparation*
- *Example Configuration*
- *Running the Demo*
- *Known Issues*

Overview The dsp_xaf_record application demonstrates audio processing using the DSP core, the Xtensa Audio Framework (XAF) middleware library, with a focus on audio recording, processing and voice recognition (VIT - Voice Intelligent Technology).

As shown in the table below, the application is supported on several development boards and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Functionality The application includes the following main components:

1. **ARM Core (CM33)** - Handles user interface and communicates with the DSP core
2. **DSP Core** - Processes audio data using the Xtensa Audio Framework (XAF)

The typical audio processing pipeline includes:

- Audio source component - DMIC audio
- VIT component (perform voice recognition)
- Renderer component (playback on codec)

The application demonstrates recording from digital microphones (DMIC), processing the audio with voice enhancement algorithms, performing voice recognition, and prints back in console detected WakeWord and list of commands.

Hardware Requirements

- Development board (one of the following):
  - EVK-MIMXRT595 board
  - EVK-MIMXRT685 board
  - MIMXRT685-AUD-EVK board

  - MIMXRT700-EVK board
- Micro USB cable
- JTAG/SWD debugger
- Headphones with 3.5 mm stereo jack
- Personal Computer

Hardware Modifications Some development boards need some hardware modifications to run the application.

- *EVK-MIMXRT595:*

  To enable the example audio using WM8904 codec, connect pins as follows:

  - JP7-1 <–> JP8-2

  Note: The I3C Pin configuration in pin_mux.c is verified for default 1.8V, for 3.3V, need to manually configure slew rate to slow mode for I3C-SCL/SDA.

- *EVK-MIMXRT685:*

  To enable the example audio using WM8904 codec, connect pins as follows:

  - JP7-1 <–> JP8-2

- *MIMXRT685-AUD-EVK*

  1. Set the hardware jumpers (Tower system/base module) to default settings.
  2. Set hardware jumpers JP2 2<–>3, JP44 1<–>2 and JP45 1<–>2.

- *MIMXRT700-EVK:*

  Set the hardware jumpers to default settings.

Preparation

1. Connect headphones to Audio HP / Line-Out connector.
   - EVK-MIMXRT595 - J4
   - EVK-MIMXRT685 - J4
   - MIMXRT685-AUD-EVK - J4, J50 for third channel when using 3 microphones
   - MIMXRT700-EVK - J29
2. Connect a micro USB cable between the PC host and the debug USB port on the development board.
   - EVK-MIMXRT595 - J40
   - EVK-MIMXRT685 - J5
   - MIMXRT685-AUD-EVK - J5
   - MIMXRT700-EVK - J54
3. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control

4. Download the program for CM33 core to the target board.
5. Launch the debugger in your IDE to begin running the demo.
6. If building release configuration, start the xt-ocd daemon and download the program for DSP core to the target board. If building debug configuration, launch the Xtensa IDE or xt-gdb debugger to begin running the demo.

Notes:

- DSP image can only be debugged using J-Link debugger. See the document 'Getting Started with Xplorer' for your particular board for more information.

Example Configuration The example can be configured by user. Before configuration, please check the *table* to see if the feature is supported on the development board.

Running the Demo The ARM application will power and clock the DSP, so it must be loaded prior to loading the DSP application. The DSP application can be built by the following tools: Xtensa Xplorer or Xtensa C Compiler. Application for Cortex-M33 can be built by the other toolchains listed in MCUXpresso SDK Release Notes.

The release configurations of the demo will combine both applications into one ARM image. With this, the ARM core will load and start the DSP application on startup. Pre-compiled DSP binary images are provided under dsp/binary/ directory. If you make changes to the DSP application in release configuration, rebuild ARM application after building the DSP application. If you plan to use MCUXpresso IDE for cm33 you will have to make sure that the preprocessor symbol DSP_IMAGE_COPY_TO_RAM, found in IDE project settings, is defined to the value 1 when building release configuration.

The debug configurations will build two separate applications that need to be loaded independently. DSP application can be built by the following tools: Xtensa Xplorer or Xtensa C Compiler. Required tool versions can be found in MCUXpresso SDK Release Notes for the board. Application for cm33 can be built by the other toolchains listed there. If you plan to use MCUXpresso IDE for cm33 you will have to make sure that the preprocessor symbol DSP_IMAGE_COPY_TO_RAM, found in IDE project settings, is defined to the value 0 when building debug configuration. The ARM application will power and clock the DSP, so it must be loaded prior to loading the DSP application.

In order to debug both the Cortex-M33 and DSP side of the application, please follow the instructions:

1. It is necessary to run the Cortex-M33 side first and stop the application before the DSP_Start function
2. Run the xt-ocd daemon with proper settings
3. Download and debug the DSP application

In order to get TRACE debug output from the XAF it is necessary to define XF_TRACE 1 in the project settings. It is possible to save the TRACE output into RAM using DUMP_TRACE_TO_BUF 1 define on project level. Please see the initialization of the TRACE function in the xaf_main_dsp.c file. For more details see XAF documentation.

Running on CM33

When the demo runs successfully, the CM33 terminal will display the following output (example from MIMXRT700-EVK):

```
******************************
DSP audio framework demo start
******************************
```

(continues on next page)

(continued from previous page)

```
[CM33 Main] Configure codec

[DSP_Main] Cadence Xtensa Audio Framework
[DSP_Main] Library Name    : Audio Framework (Hostless)
[DSP_Main] Library Version : 3.5
[DSP_Main] API Version     : 3.2

[DSP_Main] start
[DSP_Main] established RPMsg link
[CM33 Main] DSP image copied to DSP TCM
[CM33 Main][APP_DSP_IPC_Task] start
[CM33 Main][APP_Shell_Task] start

Copyright  2024  NXP

>>
```

Type help to see the command list. Similar description will be displayed on serial console (example from MIMXRT700-EVK):

```
"help": List all the registered commands

"exit": Exit program

"version": Query DSP for component versions

"record_dmic": Record DMIC audio , perform voice recognition (VIT) and playback on codec
USAGE: record_dmic [language]
For voice recognition say supported WakeWord and in 3s frame supported command.
If selected model contains strings, then WakeWord and list of commands will be printed in console.
NOTE: this command does not return to the shell
```

After running the "record_dmic en" command, similar output will be printed

```
[CM33 CMD] Setting VIT language to en
[DSP_Main] Number of channels 1, sampling rate 16000, PCM width 32
[CM33 CMD] [APP_DSP_IPC_Task] response from DSP, cmd: 13, error: 0
[DSP Record] Audio Device Ready
[CM33 CMD] DSP DMIC Recording started
[CM33 CMD] To see VIT functionality say wakeword and command
[DSP VIT] VIT Model info
[DSP VIT]   VIT Model Release = 0x40a00
[DSP VIT]   Language supported : English
[DSP VIT]   Number of WakeWords supported : 2
[DSP VIT]   Number of Commands supported : 12
[DSP VIT]   VIT_Model integrating WakeWord and Voice Commands strings : YES
[DSP VIT]   WakeWords supported :
[DSP VIT]    'HEY NXP'
[DSP VIT]    'HEY TV'
[DSP VIT]   Voice commands supported :
[DSP VIT]    'MUTE'
[DSP VIT]    'NEXT'
[DSP VIT]    'SKIP'
[DSP VIT]    'PAIR DEVICE'
[DSP VIT]    'PAUSE'
[DSP VIT]    'STOP'
[DSP VIT]    'POWER OFF'
[DSP VIT]    'POWER ON'
[DSP VIT]    'PLAY MUSIC'
[DSP VIT]    'PLAY GAME'
[DSP VIT]    'WATCH CARTOON'
```

(continues on next page)

(continued from previous page)

```
[DSP VIT]    'WATCH MOVIE'
[DSP Record] connected CAPTURER -> GAIN_0
[DSP Record] connected XA_GAIN_0 -> XA_VIT_PRE_PROC_0
[DSP Record] connected XA_VIT_PRE_PROC_0 -> XA_RENDERER_0
[DSP VIT]  - WakeWord detected 1 HEY NXP
[DSP VIT]  - Voice Command detected 6 STOP
```

Xtensa IDE log of successful start of command:

```
Number of channels 1, sampling rate 16000, PCM width 16
Audio Device Ready
connected CAPTURER -> GAIN_0
connected CAPTURER -> XA_VIT_PRE_PROC_0
connected XA_VIT_PRE_PROC_0 -> XA_RENDERER_0
```

Running on DSP

Debug configuration: When the demo runs successfully, the terminal will display the following:

```
Cadence Xtensa Audio Framework
  Library Name    : Audio Framework (Hostless)
  Library Version : 3.2
  API Version     : 3.0

[DSP_Main] start
[DSP_Main] established RPMsg link
Number of channels 1, sampling rate 16000, PCM width 16

connected CAPTURER -> GAIN_0
connected XA_GAIN_0 -> XA_VIT_PRE_PROC_0
connected XA_VIT_PRE_PROC_0 -> XA_RENDERER_0
```

Known Issues There are limited features in release SRAM target because of memory limitations. To enable/disable components, set appropriate preprocessor define in project settings to 0/1 (e.g. XA_VIT_PRE_PROC etc.). Debug and flash targets have full functionality enabled.

XAF USB Example

Table of Content

- *Overview*
- *Functionality*
- *Hardware Requirements*
- *Hardware Modifications*
- *Preparation*
- *Running the Demo*
- *Known Issues*

Overview The dsp_xaf_usb_demo application demonstrates audio processing using the DSP core, the Xtensa Audio Framework (XAF) middleware library.

As shown in the table below, the application is supported on several development boards and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* section before running the demo.

Functionality The application includes the following main components:

1. **ARM Core (CM33)** - Handles user interface, and communicates with the DSP core
2. **DSP Core** - Processes audio data using the Xtensa Audio Framework (XAF)

The XAF USB example demonstrates DSP-powered USB audio processing in two configurations: USB speaker and USB microphone. The application uses shell commands to switch between modes, with the ARM core handling USB communication while the DSP processes audio.

- USB Speaker Mode (USB2.0 Line out): Receives audio from a USB host, processes it on the DSP, and outputs through the headphone jack, making the device function as a USB speaker for your computer.
- USB Microphone Mode (DMIC USB2.0): Captures audio from the onboard digital microphones, processes it on the DSP, and streams it to a USB host as a standard audio input device.

Hardware Requirements

- Development board (one of the following):
  - EVK-MIMXRT595 board
  - EVK-MIMXRT685 board
  - MIMXRT685-AUD-EVK board
  - MIMXRT700-EVK board
- 2x Micro USB cable
- JTAG/SWD debugger
- Headphones with 3.5 mm stereo jack
- Personal Computer

Hardware Modifications Some development boards need some hardware modifications to run the application.

- *EVK-MIMXRT595:*

  To enable the example audio using WM8904 codec, connect pins as follows:

  - JP7-1 <–> JP8-2

  Note: The I3C Pin configuration in pin_mux.c is verified for default 1.8V, for 3.3V, need to manually configure slew rate to slow mode for I3C-SCL/SDA.

- *EVK-MIMXRT685:*

  To enable the example audio using WM8904 codec, connect pins as follows:

  - JP7-1 <–> JP8-2

- *MIMXRT685-AUD-EVK*
  - Set the hardware jumpers (Tower system/base module) to default settings.
  - Set hardware jumpers JP2 2<–>3, JP44 1<–>2 and JP45 1<–>2.

- *MIMXRT700-EVK:*

  Set the hardware jumpers to default settings.

Preparation

1. Connect headphones to Audio HP / Line-Out connector.
   - EVK-MIMXRT595 - J4
   - EVK-MIMXRT685 - J4
   - MIMXRT685-AUD-EVK - J4
   - MIMXRT700-EVK - J29
2. Connect the first micro USB cable between the PC host and the debug USB port on the development board.
   - EVK-MIMXRT595 - J40
   - EVK-MIMXRT685 - J5
   - MIMXRT685-AUD-EVK - J5
   - MIMXRT700-EVK - J54
3. Connect the second micro USB cable between the PC host and the USB port on the development board.
   - EVK-MIMXRT595 - J38
   - EVK-MIMXRT685 - J7
   - MIMXRT685-AUD-EVK - J7
   - MIMXRT700-EVK - J40
4. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
5. Download the program for CM33 core to the target board.
6. Launch the debugger in your IDE to begin running the demo.
7. If building release configuration, start the xt-ocd daemon and download the program for DSP core to the target board. If building debug configuration, launch the Xtensa IDE or xt-gdb debugger to begin running the demo.

Notes:

- DSP image can only be debugged using J-Link debugger. See the document 'Getting Started with Xplorer' for your particular board for more information.

Running the Demo The ARM application will power and clock the DSP, so it must be loaded prior to loading the DSP application. The DSP application can be built by the following tools: Xtensa Xplorer or Xtensa C Compiler. Application for Cortex-M33 can be built by the other toolchains listed in MCUXpresso SDK Release Notes.

The release configurations of the demo will combine both applications into one ARM image. With this, the ARM core will load and start the DSP application on startup. Pre-compiled DSP

binary images are provided under dsp/binary/ directory. If you make changes to the DSP application in release configuration, rebuild ARM application after building the DSP application. If you plan to use MCUXpresso IDE for cm33 you will have to make sure that the preprocessor symbol DSP_IMAGE_COPY_TO_RAM, found in IDE project settings, is defined to the value 1 when building release configuration.

The debug configurations will build two separate applications that need to be loaded independently. DSP application can be built by the following tools: Xtensa Xplorer or Xtensa C Compiler. Required tool versions can be found in MCUXpresso SDK Release Notes for the board. Application for cm33 can be built by the other toolchains listed there. If you plan to use MCUXpresso IDE for cm33 you will have to make sure that the preprocessor symbol DSP_IMAGE_COPY_TO_RAM, found in IDE project settings, is defined to the value 0 when building debug configuration. The ARM application will power and clock the DSP, so it must be loaded prior to loading the DSP application.

In order to debug both the Cortex-M33 and DSP side of the application, please follow the instructions:

1. It is necessary to run the Cortex-M33 side first and stop the application before the DSP_Start function
2. Run the xt-ocd daemon with proper settings
3. Download and debug the DSP application

In order to get TRACE debug output from the XAF it is necessary to define XF_TRACE 1 in the project settings. It is possible to save the TRACE output into RAM using DUMP_TRACE_TO_BUF 1 define on project level. Please see the initialization of the TRACE function in the xaf_main_dsp.c file. For more details see XAF documentation.

Running on CM33

When the demo runs successfully, the CM33 terminal will display the following output (example from MIMXRT700-EVK):

```
******************************
DSP audio framework demo start
******************************

[CM33 Main] Configure codec

[DSP_Main] Cadence Xtensa Audio Framework
[DSP_Main] Library Name    : Audio Framework (Hostless)
[DSP_Main] Library Version : 3.5
[DSP_Main] API Version     : 3.2

[DSP_Main] start
[DSP_Main] established RPMsg link
[CM33 Main] DSP image copied to DSP TCM
[CM33 Main][APP_DSP_IPC_Task] start
[CM33 Main][APP_Shell_Task] start

Copyright 2024 NXP

>>
```

Type help to see the command list. Similar description will be displayed on serial console (example from MIMXRT700-EVK):

```
"help": List all the registered commands

"exit": Exit program
```

(continues on next page)

(continued from previous page)

```
  "version": Query DSP for component versions

  "usb_speaker": Perform usb speaker device and playback on DSP
     USAGE: usb_speaker [start|stop]
     start          Start usb speaker device and playback on DSP
     stop           Stop usb speaker device and playback on DSP

  "usb_mic": Record DMIC audio and playback on usb microphone audio device
     USAGE: usb_mic [start|stop]
     start          Start record and playback on usb microphone audio device
     stop           Stop record and playback on usb microphone audio device
```

When usb_speaker command starts playback successfully, the terminal will display following output:

```
[APP_DSP_IPC_Task] response from DSP, cmd: 21, error: 0
DSP USB playback start
>>
```

Xtensa IDE log when command is playing a file:

```
USB speaker start, initial buffer size: 960
[DSP_USB_SPEAKER] Audio Device Ready
[DSP_USB_SPEAKER] post-proc/pcm_gain component started
[DSP_USB_SPEAKER] post-proc/client_proxy component started
[DSP_USB_SPEAKER] Connected post-proc/pcm_gain -> post-proc/client_proxy
[DSP_USB_SPEAKER] renderer component started
[DSP_USB_SPEAKER] Connected post-proc/client_proxy -> renderer
[DSP_ProcessThread] start
[DSP_BufferThread] start
[DSP_CleanupThread] start
```

The USB device on your host will be enumerated as XAF USB DEMO.

Xtensa IDE will not show any additional log entry.

Running the demo DSP

Debug configuration: When the demo runs successfully, the terminal will display the following:

```
Cadence Xtensa Audio Framework
  Library Name    : Audio Framework (Hostless)
  Library Version : 2.6p1
  API Version     : 2.0

[DSP_Main] start
[DSP_Main] established RPMsg link
```

Known Issues

- When starting the “usb_speaker” after the “usb_mic” command, the sound output may be distorted. Please power cycle the board.

3.8 Wireless

3.8.1 NXP Wireless Framework and Stacks

Wi-Fi, Bluetooth, 802.15.4

Application notes

- Link AN12918-Wi-Fi-Tx-Power-Table-and-Channel-Scan-Management-for-i.MX-RT-SDK.pdf
- Link TN00066-WFA-Derivative-Certification-Process.pdf

User manuals

- Link UM11441-Getting-Started-with-NXP-based-Wireless-Modules-and-i.MX-RT-Platforms.pdf
- UM11442-NXP-Wi-Fi-and-Bluetooth-Demo-Applications-for-i.MX-RT-Platforms.pdf
- Link UM11443-NXP-Wi-Fi-and-Bluetooth-Debug-Feature-Configuration-Guide-for-i.MX-RT-Platforms.pdf
- Link UM11567-WFA-Certification-Guide-for-NXP-based-Wireless-Modules-on-i.MX-RT-Platform-Running-RTOS.pdf

Release notes Wireless SoC features and release notes for FreeRTOS

About this document

This document provides information about the supported features, release versions, fixed and/or known issues, performance of the Wi-Fi, Bluetooth/802.15.4 radios, including the coexistence.

The SDK release version 26.09.00 has been tested for the wireless SoCs listed in Supported products.

Supported products

- 88W8987
- IW416
- IW611
- IW612
- AW611
- RW610
- RW612

Parent topic: *About this document*

Features

Wi-Fi radio

Client mode

Features	Sub features
802.11n - High throughput	2.4 GHz band operation supported channel bandwidth: 20 M
802.11n - High throughput	2.4 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 20 MHz

Table 52 – continued from

Features	Sub features
802.11n - High throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	Short/long guard interval (400 ns/800 ns)
802.11n - High throughput	Data rates up to 72 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	Data rates up to 150 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	1 spatial stream (1x1)
802.11n - High throughput	HT protection mechanisms
802.11n - High throughput	Aggregated MAC protocol data unit (AMPDU) TX and RX sup
802.11n - High throughput	Aggregated MAC service data unit (AMSDU) 4k TX and RX su
802.11n - High throughput	TX MCS rate adaptation (BGN)
802.11n - High throughput	RX low density parity check (LDPC) 1x1 20 MHz and 40 MHz
802.11n - High throughput	HT Beamformee (explicit)
802.11ac - Very high throughput	2.4 GHz band supported channel bandwidth: 20MHz
802.11ac - Very high throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11ac - Very high throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11ac - Very high throughput	5 GHz band supported channel bandwidth: 80 MHz
802.11ac - Very high throughput	Data rates up to 86.7 Mbps (MCS0 to MCS 8)
802.11ac - Very high throughput	Data rates up to 433.3 Mbps (MCS 0 to MCS 9) - 1x1
802.11ac - Very high throughput	MU-MIMO Beamformee (Explicit and Implicit)
802.11ac - Very high throughput	RTS/CTS with BW signaling
802.11ac - Very high throughput	Operation mode notification
802.11ac - Very high throughput	Backward compatibility with non-VHT devices
802.11ac - Very high throughput	TX VHT MCS rate adaptation
802.11ac - Very high throughput	Low density parity check (LDPC)
802.11ax - High efficiency	2.4 GHz band supported channel bandwidth: 20MHz
802.11ax - High efficiency	5 GHz band supported channel bandwidth: 20 MHz
802.11ax - High efficiency	5 GHz band supported channel bandwidth: 40 MHz
802.11ax - High efficiency	5 GHz band supported channel bandwidths: 80 MHz
802.11ax - High efficiency	OFDMA (UL/DL, 106 RU)
802.11ax - High efficiency	OFDMA (UL/DL, 484 RU)
802.11ax - High efficiency	1024 QAM
802.11ax - High efficiency	Target wake time (TWT)
802.11ax - High efficiency	256 QAM modulation – MCS8 and MCS9
802.11ax - High efficiency	1024 QAM modulation – MCS10 and MCS11, 2.4 GHz
802.11ax - High efficiency	1024 QAM modulation – MCS10 and MCS11, 5 GHz
802.11ax - High efficiency	DCM
802.11ax - High efficiency	DCM
802.11ax - High efficiency	ER (extended range)
802.11ax - High efficiency	SU Beamforming
802.11ax - High efficiency	OMI (operating mode indication)
802.11a/b/g features	802.11b/g data rates up to 54 Mbit/s
802.11a/b/g features	802.11a data rates up to 54 Mbit/s
802.11a/b/g features	TX rate adaptation (BG)
802.11a/b/g features	Fragmentation/defragmentation
802.11a/b/g features	ERP protection, slot time, preamble
802.11d	802.11d - Regulatory domain/operating class/country info
802.11e QoS	EDCA [enhanced distributed channel access] / WMM (wirele
802.11i security	Opensource WPA Supplicant Support
802.11i security	WPA2-PSK AES | WPA Supplicant
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | WPA
802.11i security	WPA2+WPA3 PSK Mixed Mode (WPA3 Transition Mode) | W
802.11i security	Wi-Fi Enhanced Open - OWE (Opportunistic Wireless Encry
802.11i security	802.1x EAP Authentication Methods3 | WPA Supplicant
802.11i security	WPA2-Enterprise Mixed Mode | WPA Supplicant
802.11i security	WPA3-Enterprise3 (Suite-B) |National Security Algorithm (C
802.11i security	802.11w - PMF (Protected Management Frames) | WPA Supp

Table 52 – continued from

Features	Sub features
802.11i security	Embedded Supplicant Support
802.11i security	WPA2-PSK AES | Embedded Supplicant
802.11i security	WPA+WPA2 PSK Mixed Mode | Embedded Supplicant
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | Embe
802.11i security	802.11w - PMF (Protected Management Frames) | Embedde
802.11i security	Wi-Fi Roaming
802.11i security	WPA3 Enterprise3
Power save mode	Deep sleep
Power save mode	IEEE power save
Power save mode	Host sleep/WoWLAN (inband)3
Power save mode	Host sleep/WoWLAN (outband)3
Power save mode	U-APSD
802.11w - PMF (protected management frames)	PMF require and capable
802.11w - PMF (protected management frames)	Unicast management frames - Encryption/decryption - usin
802.11w - PMF (protected management frames)	Broadcast management frames - Encryption/decryption - us
802.11w - PMF (protected management frames)	SA query request/response
802.11w - PMF (protected management frames)	PMF support using embedded supplicant
DPP functionality	Wi-Fi easy connect3
General features	Embedded supplicant
General features	Host sleep packet filtering
General features	Host-based supplicant
General features	Embedded MLME
General features	EDMAC - EU adaptivity support (ETSI certification)
General features	External coexistence
General features	IPv6 NS offload
General features	FIPS
General features	TKIP1
General features	RF test mode
General features	802.11k
General features	802.11v
General features	DFS radar detection in peripheral mode (follow AP)5
General features	Embedded roaming based on RSSI threshold beacon loss
General features	ARP offload
General features	Cloud keep alive
General features	UNII-4 channel support
General features	ClockSync using TSF
General features	Auto reconnect
General features	CSI (channel state information)
General features	Ambient Motion Index (AMI)3
General features	Independent reset (in-band)3
General features	Independent reset (out-band)3
General features	Wi-Fi agile multiband
General features	Network co-processor (NCP) mode
General features	802.11mc - WLS (Wi-Fi location service)3
General features	802.11az3
General features	WPA3 Transition Disable

Parent topic: Wi-Fi radio

[1] As per Wi-Fi specification, connecting in TKIP security in non 802.11n mode is allowed.

[2] Support available in host-base supplicant.

[3] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature.

[4] Read more about NCP feature in *References*. [5] To enable the feature, CONFIG_ECSA = 1 must

be defined in wifi_config.h (does not apply to RW610 and RW612).

AP mode

Features	Sub features
802.11n - High throughput	2.4 GHz band operation supported channel bandwidth: 20 M
802.11n - High throughput	2.4 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	Short/long guard interval (400 ns/800 ns)
802.11n - High throughput	Data rates up to 72 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	Data rates up to 150 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	1 spatial stream (1x1)
802.11n - High throughput	HT protection mechanisms
802.11n - High throughput	Aggregated MAC protocol data unit (AMPDU) Rx support
802.11n - High throughput	Aggregated MAC service data unit (AMSDU) -4k RX support
802.11n - High throughput	Max client support (up to 8 devices)
802.11n - High throughput	TX MCS rate adaptation (BGN)
802.11n - High throughput	RX low density parity check (LDPC)
802.11ac – Very high throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11ac – Very high throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11ac – Very high throughput	5 GHz band supported channel bandwidth: 80MHz
802.11ac – Very high throughput	Short/long guard interval (400ns/800ns)
802.11ac – Very high throughput	Data rates up to 86.7 Mbps (MCS0 to MCS 8)
802.11ac – Very high throughput	Data rates up to 433.3 Mbps (MCS 0 to MCS 9)
802.11ac – Very high throughput	Single user- Aggregated MAC protocol data unit (SU-AMPDU
802.11ac – Very high throughput	RTS/CTS with BW signaling
802.11ac – Very high throughput	Backward compatibility with non-VHT devices
802.11ac – Very high throughput	TX VHT MCS rate adaptation
802.11ac – Very high throughput	MU-MIMO Beamformee (explicit and implicit)
802.11ac – Very high throughput	Operation mode notification
802.11ax – High efficiency	2.4 GHz band operation (20 MHz channel bandwidth)
802.11ax – High efficiency	2.4 GHz band operation (40 MHz channel bandwidth)
802.11ax – High efficiency	5 GHz band operation (20MHz channel bandwidth)
802.11ax – High efficiency	5 GHz band operation (40MHz channel bandwidth)
802.11ax – High efficiency	5 GHz band operation (80 MHz channel bandwidth)
802.11d	802.11d - Regulatory domain/operating class/country info
802.11e -QoS	EDCA [enhanced distributed channel access] / WMM (wirele
802.11i security	Hostapd Support
802.11i security	WPA2-PSK AES | hostapd
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | Hosta
802.11i security	WPA2+WPA3 PSK Mixed Mode (WPA3 Transition Mode) | H
802.11i security	Wi-Fi Enhanced Open - OWE (Opportunistic Wireless Encry
802.11i security	802.1x EAP Authentication Methods | Hostapd
802.11i security	WPA2-Enterprise Mixed Mode1 | Hostapd
802.11i security	WPA3-Enterprise (Suite-B)1 |National Security Algorithm (C
802.11i security	802.11w - PMF (Protected Management Frames) | Hostapd
802.11i security	Embedded Authenticator
802.11i security	WPA2-PSK AES | Embedded Supplicant
802.11i security	WPA+WPA2 PSK Mixed Mode | Embedded Supplicant
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | Embe
802.11i security	802.11w - PMF (Protected Management Frames) | Embedde
802.11y	Extended channel switch announcement (ECSA)
802.11w - protected management frames (PMF)	PMF require and capable
802.11w - protected management frames (PMF)	Unicast management frames -Encryption/decryption - using
802.11w - protected management frames (PMF)	Broadcast management frames -encryption/decryption - usi

Table 53 – continued from prev

Features	Sub features
802.11w - protected management frames (PMF)	SA query request/response
General features	Embedded authenticator
General features	Embedded MLME
General features	EU adaptivity support
General features	Automatic channel selection (ACS)
General features	External coexistence (software interface)
General features	Independent reset (in-band)1
General features	Network co-processor (NCP) mode2
General features	Vendor specific IE (custom IE)
General features	Hidden SSID (broadcast SSID disabled)
General features	MAC address filter
General features	Multiple external STA support
General features	WPA3 Transition Disable

Parent topic: Wi-Fi radio

[1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory. [2] Read more about NCP feature in *References*.

AP-STA mode

Features	Sub features	88W8	IW41	IW611/IV	RW610/R\	IW61	AW611
Simultaneous AP-STA operation (same channel)	AP-STA functionality	Y	Y	Y	Y	Y	Y
SAD	Software antenna diversity1	Y	Y	Y	Y	Y	Y

Parent topic: Wi-Fi radio

[1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature.

Parent topic: *Features*

Wi-Fi Generic features

Features	Sub features	88W898	IW41	IW611/IW	RW610/RW	IW61	AW611
Generi	Firmware download (parallel)1	Y	Y	Y	N	N	Y
Generi	Secure boot	N	N	Y	Y	Y	Y
Generi	Kconfig memory optimizer3	Y	Y	Y	Y	Y	Y
Generi	Firmware Compression2	N	Y	N	N	N	N
Generi	u-AP intra-BSS	Y	N	Y	Y	Y	Y
Generi	Net Monitor Mode	N	N	N	Y	Y	N
Generi	Net Monitor Mode with packet transmission	N	N	N	Y	Y	N
Generi	In-Channel Net Monitor mode	N	N	N	N	N	N

Parent topic: Wi-Fi radio

[1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature. [2] The feature is used to compress the Wi-Fi Bluetooth firmware and optimize the flashing of the host [3] Refer to *10*.

Wi-Fi direct/P2P

Features	Sub features	88W89	IW41	IW611/IW	RW610/RW	IW61	AW6113
P2P basic functionality1	P2P Auto GO	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P GO	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P GC	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P Persistent Group	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P Invitation	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P Device Discovery	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P Provision Discovery	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P simultaneous GO + STA	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P simultaneous GC + uAP	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P same channel simultaneous GO + uAP	Y	Y	Y	Y	Y	Y

Parent topic: Wi-Fi radio

[1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature. [2] This is an experimental software release for this feature for IW416. [3] Contact your support representative to use this feature for.

Bluetooth radio

Bluetooth classic

Feature	Sub feature	88W8	IW41	IW611/	RW610/	IW61	AW611
General features	Bluetooth Class 1.5 and Class 2 support	Y	Y	Y	N	N	Y
General features	Scatternet support	Y	Y	Y	N	N	Y
General features	Maximum of seven simultaneous ACL connections – Central links	Y	Y	Y	N	N	Y
General features	Automatic packet type selection	Y	Y	Y	N	N	Y
General features	Bluetooth - 2.1 to 5.0 specification support	Y	Y	Y	N	N	Y
General features	Low power sniff	Y	Y	Y	N	N	Y
General features	Deep sleep using out-of-band	Y	Y	N	N	N	N
General features	Wake on Bluetooth (SoC to host)	Y	Y	Y	N	N	Y
General features	Independent reset (in-band)1	Y	Y	Y	Y	N	Y
General features	Independent reset (out-band)1	Y	Y	N	N	N	N
General features	Firmware download (parallel)1	Y	Y	N	N	N	N
General features	RF test mode	Y	Y	Y	N	N	Y
Bluetooth packet type supported	ACL (DM1, DH1, DM3, DH3, DM5, DH5, 2-DH1, 2-DH3, 2-DH5, 3-DH1, 3-DH3, 3-DH5)	Y	Y	Y	N	N	Y
Bluetooth packet type supported	SCO (HV1, HV3)	Y	Y	Y	N	N	Y
Bluetooth packet type supported	eSCO (EV3, EV4, EV5, 2EV3, 3EV3, 2EV5, 3EV5)	Y	Y	Y	N	N	Y
Bluetooth profiles supported	A2DP source/sink	Y	Y	Y	N	N	Y
Bluetooth profiles supported	AVRCP target/controller	Y	Y	Y	N	N	Y
Bluetooth profiles supported	HFP Dev/AG	Y	Y	Y	N	N	Y
Bluetooth profiles supported	OPP server/client	Y	Y	Y	N	N	Y
Bluetooth profiles supported	SPP server/client	Y	Y	Y	N	N	Y
Bluetooth profiles supported	HID target/device	Y	Y	Y	N	N	Y
Bluetooth audio features	PCM NBS central/peripheral	Y	Y	Y	N	N	Y
Bluetooth audio features	PCM WBS central/peripheral	Y	Y	Y	N	N	Y

Parent topic: Bluetooth radio

[1] Experimental feature intended for evaluation/early development only and not production. Incomplete mandatory certification.

Bluetooth LE

Features	Sub features
Generic features	Maximum 16 Bluetooth LE connections (central role)
Generic features	Deep sleep using out-of-band
Generic features	Wake on Bluetooth LE (SoC to Host)
Generic features	RF Test mode
Bluetooth profile support	Bluetooth LE GATT
Bluetooth profile support	Bluetooth LE HID over GATT
Bluetooth profile support	Bluetooth LE GAP
Bluetooth LE 4.0 support	Low Energy physical layer
Bluetooth LE 4.0 support	Low Energy link layer
Bluetooth LE 4.0 support	Enhancements to HCI for Low Energy
Bluetooth LE 4.0 support	Low energy direct test mode
Bluetooth 4.1 support	Low duty cycle directed advertising
Bluetooth 4.1 support	Bluetooth LE dual mode topology
Bluetooth 4.1 support	Bluetooth LE privacy v1.1
Bluetooth 4.1 support	Bluetooth LE link layer topology
Bluetooth 4.2 support	Bluetooth LE secure connection
Bluetooth 4.2 support	Bluetooth LE link layer privacy v1.2
Bluetooth 4.2 support	Bluetooth LE data length extension
Bluetooth 4.2 support	Link layer extended scanner filter policies
Bluetooth 5.0 support	Bluetooth LE 2 Mbps support
Bluetooth 5.0 support	High duty cycle directed advertising
Bluetooth 5.0 support	Low Energy advertising extension
Bluetooth 5.0 support	Low Energy long range
Bluetooth 5.0 support	Low Energy periodic advertisement
Bluetooth 5.2 support	Low Energy power control
Bluetooth LE audio support1 2	Isochronous channel
Bluetooth LE audio support1 2	Broadcast LE Audio BIS source
Bluetooth LE audio support1 2	Broadcast LE Audio BIS sink
Bluetooth LE audio support1 2	Broadcast LE Audio BIG Validation
Bluetooth LE audio support1 2	Broadcast LE Audio Phy: 1M/2M/ coded
Bluetooth LE audio support1 2	Broadcast LE Audio framed mode
Bluetooth LE audio support1 2	Broadcast LE Audio unframed mode
Bluetooth LE audio support1 2	Broadcast LE Audio sequential packing
Bluetooth LE audio support1 2	Broadcast LE Audio: Mono and Stereo
Bluetooth LE audio support1 2	Broadcast LE Audio BIS encrypted audio
Bluetooth LE audio support1 2	Broadcast LE Audio BIS unencrypted audio
Bluetooth LE audio support1 2	Unicast LE Audio CIS source
Bluetooth LE audio support1 2	Unicast LE Audio CIS sink
Bluetooth LE audio support1 2	Unicast LE Audio CIG validation
Bluetooth LE audio support1 2	Unicast LE Audio CIS synchronization
Bluetooth LE audio support1 2	Unicast LE Audio Phy: 1M/2M/ coded
Bluetooth LE audio support1 2	Unicast LE Audio framed mode
Bluetooth LE audio support1 2	Unicast LE Audio unframed mode
Bluetooth LE audio support1 2	Unicast LE Audio sequential packing
Bluetooth LE audio support1 2	Unicast LE Audio: mono and stereo
Bluetooth LE audio support1 2	Unicast LE Audio CIS encrypted audio
Bluetooth LE audio support1 2	Unicast LE Audio CIS unencrypted audio
Bluetooth LE audio support1 2	Unicast LE Audio TX/RX and bidirectional traffic

Table 54 – continued from pre

Features	Sub features
Bluetooth LE audio support1 2	ISO interval for LE Audio: 7.5ms 10ms 20ms 30ms
Bluetooth LE audio support1 2	Sampling frequency for LE Audio: 8kHz 16kHz 24kHz, 32kH
Bluetooth LE audio support1 2	LE Audio Auracast use cases: Auracast streaming 2 BISes
Bluetooth LE audio support1 2	LE Audio Unicast use cases: Unicast streaming 2 CISes
Bluetooth LE audio support1 2	LE Audio Unicast Use cases: Unicast streaming 4 CISes
Bluetooth LE audio support1 2	A2DP + Auracast/Unicast Bridge use cases – CIS/BIS
BCA TDM Coexistence mode (shared antenna)	STA + Bluetooth coexistence
BCA TDM Coexistence mode (shared antenna)	STA + Bluetooth LE coexistence
BCA TDM Coexistence mode (shared antenna)	STA + Bluetooth + Bluetooth LE coexistence
BCA TDM Coexistence mode (shared antenna)	AP + Bluetooth coexistence
BCA TDM Coexistence mode (shared antenna)	AP + Bluetooth LE coexistence
BCA TDM Coexistence mode (shared antenna)	AP + Bluetooth + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	STA + Bluetooth coexistence
BCA TDM coexistence mode (separate antenna)	STA + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	STA + Bluetooth + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	AP + Bluetooth coexistence
BCA TDM coexistence mode (separate antenna)	AP + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	AP + Bluetooth + Bluetooth LE coexistence

Note: Details of the tested Bluetooth LE Audio use cases:

- Number of streams:
  - 1-CIG | upto 4-CIS with 1 LE ACL (for 4-CIS: execute only mono UCs, SDU Int: 10ms)
  - 1-CIG | upto 4-CIS with 4 separate LE ACL (for 4-CIS: SDU Size= Max 100 Oct, PHY=2M, RTN=1, SDU Int: 10ms only) (execute only mono UCs for 4-CIS)
  - 1-BIG | upto 4-BIS (for 4-BIS: execute only mono UCs, SDU Int: 10ms only)
- PHY: 2M and 1M
- Audio mode: mono (for 1 to 4 streams) and stereo (for 1 stream)
- Packing: sequential and interleaved
- Bit rate: maximum 96kbps
  - For 1-CIG with upto 3-CIS: maximum bit rate 96kbps
  - For 1-CIG with 4-CIS: maximum bit rate 80kbps
  - For 1-BIG with 4-BIS: maximum bit rate 80kbps
  - For 2-CIG cases: maximum bit rate 80kbps
- Mode: unframed mode
- 48_5 and 48_6 mono and stereo configurations are not supported.

Details of the tested Bluetooth coexistence (Bluetooth + Bluetooth LE Audio) use cases:

- Bluetooth + Bluetooth LE Audio
- A2DP + Bluetooth LE Audio bridging support
- A2DP sink link (central) -> LEA 2-CIS (SDU Int: 10ms only | A2DP only with SBC Codec | PHY: 2M)

Parent topic: Bluetooth radio

[1] Experimental feature intended for evaluation/early development only and not production. Incomplete mandatory certification.

[2] LE audio feature is supported for standalone scenarios only and not for BR/EDR and Wi-Fi co-existence scenarios such as LE audio + BR/EDR link or LE audio + Wi-Fi link. From the perspective

of NXP Edgefast Bluetooth host stack, LE audio feature can be disabled by the CONFIG_BT_AUDIO macro without impact on any other features. LE audio feature can be tested by the user, using their own supported host stack.

Parent topic: *Features*

802.15.4 radio

Features	Sub features	IW612	IW610	RW612
General features	Spinel over SPI	Y	N	N
General features	OpenThread RCP Mode implementing Thread1.3	Y	N	N
General features	802.15.4-2015 MAC/PHY as required by Thread 1.3	Y	Y	Y
General features	OpenThread Border Router (OTBR) v1.1	Y	Y	Y
General features	Direct/indirect transmission with/without ACK	Y	Y	Y
General features	802.15.4 CSL parent feature implementation	Y	Y	Y
General features	Enhanced Frame Pending	Y	Y	Y
General features	Enhanced keep alive	Y	Y	Y
General features	Router	Y	Y	Y
General features	Leader	Y	Y	Y
General features	Router Eligible End Device (REED)	Y	Y	Y
General features	End Device (FED, MED)	Y	Y	Y
Zigbee features	Coordinator	N	N	Y
Zigbee features	Router	N	N	Y
Zigbee features	End Device (RX ON)	N	N	Y
Zigbee features	R23	N	N	Y
Zigbee features	OTA Client	N	N	Y
Zigbee features	OTA server	N	N	Y
Matter features	Matter over Wi-Fi	Y	N	N
Matter features	Matter over Thread	Y	N	Y

Parent topic: *Features*

Coexistence

Wi-Fi and Bluetooth/802.15.4 coexistence

Features	Sub features	IW6´	IW6´	RW612
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	STA + Bluetooth	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Mobile AP + Bluetooth	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth LE + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + Bluetooth LE + Wi-Fi	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth LE2	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth + Bluetooth LE	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + OpenThread + Wi-Fi	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth LE + OpenThread + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + Bluetooth LE + OpenThread + Wi-Fi	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Single antenna configuration	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	External Coexistence PTA	N	Y	Y

Parent topic: Coexistence

[1] Experimental feature intended for evaluation/early development only and not production. Incomplete mandatory certification.

[2] The narrow-band radio can be configured to support Bluetooth LE, 802.15.4, and to time-slice between Bluetooth LE and 802.15.4.

Parent topic: *Features*

Feature enable and memory impact

Features	Macros to enable the feature	Memory impact
CSI	CONFIG_CSI	Flash - 60K, RAM - 4K
AMI	CONFIG_CSI_AMI3	Flash - 2032K, RAM - 772K
DPP	CONFIG_WPA_SUPP_DPP	Flash - 240K, RAM - 12K
Independent reset	CONFIG_WIFI_IND_DNLDCONFIG_WIFI_IND_RESET	Minimal
Parallel firmware download Wi-Fi	CONFIG_WIFI_IND_DNLD	Minimal
Parallel firmware download Bluetooth	CONFIG_BT_IND_DNLD	Minimal
WPA3 enterprise	CONFIG_WPA_SUPP_CRYPTO_ENTERPRISE [Macros specific to EAP-methods included] CONFIG_EAP_TLS CONFIG_EAP_PEAP CONFIG_EAP_TTLS CONFIG_EAP_FAST CONFIG_EAP_SIM CONFIG_EAP_AKA CONFIG_EAP_AKA_PRIME	Flash - 165K, RAM - 18K
WPA2 enterprise	CONFIG_WPA_SUPP_CRYPTO_ENTERPRISE [Macros specific to EAP-methods included] CONFIG_EAP_TLS CONFIG_EAP_PEAP CONFIG_EAP_TTLS CONFIG_EAP_FAST CONFIG_EAP_SIM CONFIG_EAP_AKA CONFIG_EAP_AKA_PRIME	Flash - 165K, RAM - 18K
Host sleep	CONFIG_HOST_SLEEP	Minimal
WMM	CONFIG_WMM1	Flash - 10K, RAM - 57K
802.11mc	CONFIG_11MC CONFIG_CSI CONFIG_WLS_CSI_PROC2 CONFIG_11AZ	Flash: 52.78KB, RAM : 121.1KB
802.11az	CONFIG_11MC CONFIG_CSI[2] CONFIG_WLS_CSI_PROC2 CONFIG_11AZ	Flash: 52.78KB, RAM : 121.1KB
Non-blocking firmware download mechanism	CONFIG_FW_DNLD_ASYNC	—
Antenna diversity	CONFIG_WLAN_CALDATA_2ANT_DIVERSITY	-
P2P	CONFIG_WPA_SUPP_P2P	-

Note:

- For Wi-Fi, the macros are set with the value "**0**" by default in the file wifi_config_default.h

located in <SDK_PATH>/middleware/wifi_nxp/incl/ directory.

To enable the features, set the value of the macros to "***1****" in the file wifi_config.h located in*<SDK_Wi-Fi_Example_PATH>/ *directory****.***

- Bluetooth

  To enable the features, set the value of the macros to "**1**" in the file app_bluetooth_config.h located in <SDK_Bluetooth_Example_PATH>/ directory.

[1] The macro is not used for IW416.

[2] Prerequisite macros for 802.11mc and 802.11az features

[3] Enable PRINTF_FLOAT_ENABLE only for MCUXpresso IDE and specifically for the RT1060-EVKC and RT1170-EVKB platforms

- Go to project properties > C/C++ Build > Settings > Preprocessor.
- Add PRINTF_FLOAT_ENABLE=1

88W8987 release notes

Package information

- SDK version: 26.09.00

Parent topic: *88W8987 release notes*

Version information

- Wireless SoC: 88W8987
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 16.92.21.p156.4
  - 16 - Major revision
  - 92 - Feature pack
  - 21 - Release version
  - p156.4 - Patch number

Parent topic: *88W8987 release notes*

Host platform

- All i.MX RT platforms running FreeRTOS.
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
- Test tools
  - iPerf (version 2.1.9)

Parent topic: *88W8987 release notes*

Wi-Fi and Bluetooth certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n

- STA | 802.11ac
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA2
- STA | WPA3
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification

QDID: refer to *4*.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic: *88W8987 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- External Access Point: ASUS AX88U
- DUT: W8987 Murata (Module: **1ZM M.2**) with EVK-MIMXRT1060 EVKC platform
- DUT Power Source: External power supply
- External Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version: gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*.

Parent topic: Wi-Fi throughput

STA throughput

External APs: ASUS AX88U

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	50	50	60	62
WPA2-AES	47	49	59	62
WPA3-SAE	49	48	59	62

STA mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	62	83	121	124
WPA2-AES	61	82	120	126
WPA3-SAE	60	82	120	126

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	43	52	60	64
WPA2-AES	43	52	61	64
WPA3-SAE	43	52	60	65

STA mode throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	64	87	126	125
WPA2-AES	63	85	125	120
WPA3-SAE	63	80	125	123

STA mode throughput - AC Mode | 5 GHz Band | 20 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	48	60	73	78
WPA2-AES	47	60	73	77
WPA3-SAE	47	60	73	77

STA mode throughput - AC Mode | 5 GHz Band | 40 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	68	96	161	157
WPA2-AES	69	92	160	155
WPA3-SAE	70	94	160	155

STA mode throughput - AC Mode | 5 GHz Band | 80 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	116	111	226	233
WPA2-AES	114	99	224	200
WPA3-SAE	114	99	226	200

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple Macbook Air

Mobile AP Mode Throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	47	51	58	62
WPA2-AES	46	50	58	62
WPA3-SAE	44	50	58	62

Mobile AP Mode Throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	66	81	107	121
WPA2-AES	65	80	107	120
WPA3-SAE	65	80	108	120

Mobile AP Mode Throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	44	52	60	61
WPA2-AES	44	51	60	61
WPA3-SAE	44	51	60	61

Mobile AP Mode Throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	70	89	126	103
WPA2-AES	70	87	124	102
WPA3-SAE	70	88	125	103

Mobile AP Mode Throughput - AC Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	49	60	73	76
WPA2-AES	48	59	73	76
WPA3-SAE	48	60	73	76

Mobile AP Mode Throughput - AC Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	77	106	161	102
WPA2-AES	77	104	160	102
WPA3-SAE	77	104	160	111

Mobile AP Mode Throughput - AC Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	115	135	226	215
WPA2-AES	113	123	224	196
WPA3-SAE	113	123	225	172

Parent topic: Wi-Fi throughput

Parent topic: *88W8987 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)
- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic: *88W8987 release notes*

Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p64.1 to 16.91.21.p82

Component	Description
Wi-Fi	WPA3-R3 enabled APUT beacons does not have RSNXE when configured in H2E mode-Associated event is received even when connecting using wrong password WFA APUT Low iperf TCP/UDP Tx throughput with Realtek station

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p82 to 16.91.21.p91.6

Component	Description
Wi-Fi	In wrong password scenario, After updating new password the phone is not able to connect with DUTAP

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p91.6 to 16.91.21.p124

Component	Description
Wi-Fi	Cloud keep alive packets not seen after DUT enters host sleep. DUT is sending QOS null packets even in host sleep

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p124 to 16.91.21.p133

Component	Description
Wi-Fi	Samsung S24 Ultra and Google Pixel 7 mobiles having Android 14 are not able connect to the DUTAP with WPA3 SAE security.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p133 to 16.91.21.p142.5

Component	Description
Wi-Fi	Fails to encrypt and decrypt data with ccmp 128 and 256 using CLI crypto commands.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p142.5 to 16.91.21.p149.2

Component	Description
Wi-Fi	DUTSTA does not associate to hidden SSID beaconing in DFS channel.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p149.2 to 16.92.21.p151.7

Component	Description
Wi-Fi	Getting low TCP/UDP TP in DUT-AP 11ac-vht80 mode after hard-reset or wlan-reset.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p149.2 to 16.92.21.p151.7

Component	Description
Wi-Fi	Getting low TCP/UDP TP in DUT-AP 11ac-vht80 mode after hard-reset or wlan-reset.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p151.7 to 16.92.21.p153.5

Component	Description
Wi-Fi	Added P2P Persistance and P2P Invitation

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.5 to 16.92.21.p153.6

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.6 to 16.92.21.p153.9

Component	Description
Wi-Fi	Enabled mbedtls 3.xThe HT/VHT Wi-Fi capabilities are not being properly configured when the Device Under Test (DUT) initiates operation as a P2P Group Owner (P2P-GO).Resolved RT685 Wi-Fi EAP-TLS crash caused by expired certificates

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.9 to 16.92.21.p155.2

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p155.2 to 16.92.21.p156.1

Component	Description
Wi-Fi	Fixed roaming failure (RRM-NEIGHBOR-REP-REQUEST-FAILED) when DUT roams between APs on different 5 GHz channels on i.MXRT685 platform.
Bluetooth	Fixed same EN_RAND value being generated on every AES-CCM encryption, causing predictable random number in BT pairing.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p156.1 to 16.92.21.p156.4

Component	Description
Wi-Fi	Fixed an issue where DHCP renewal was not triggered when roaming between independent APs sharing the same SSID. Fixed an issue where the DUT failed to fall back to legacy roaming after 802.11k/11v roaming returned no usable target AP

Parent topic: Bug fixes and/or feature enhancements

Parent topic: *88W8987 release notes*

Known issues

Component	Description
NA	

Parent topic: *88W8987 release notes*

IW416 release notes

Package information

- SDK version: 26.09.00

Parent topic: *IW416 release notes*

Version information

- Wireless SoC: IW416
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 16.92.21.p166
  - 16 - Major revision
  - 92 - Feature pack
  - 21 - Release version
  - p166 - Patch number

Parent topic: *IW416 release notes*

Host platform

- All i.MX RT platforms running FreeRTOS.
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 Support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
- Test tools
  - iPerf (version 2.1.9)

Parent topic: *IW416 release notes*

Wi-Fi and Bluetooth certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA2
- STA | WPA3
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification

QDID: refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic: *IW416 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: IW416 Murata (Module: 1XK M.2) with EVK-MIMXRT1060 EVKC platform
- DUT Power Source: External power supply
- Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version: gcc-arm-none-eabi-13.2

- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*.

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	43	45	53	54
WPA2-AES	42	45	51	53
WPA3-SAE	45	42	53	53

STA mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	41	59	115	123
WPA2-AES	50	58	113	93
WPA3-SAE	28	58	114	92

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	45	48	61	56
WPA2-AES	42	48	61	44
WPA3-SAE	42	47	61	49

STA mode throughput - AN Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	43	64	118	132
WPA2-AES	50	59	118	131
WPA3-SAE	69	59	117	122

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple MacBook Air

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	43	45	53	54
WPA2-AES	42	45	51	53
WPA3-SAE	45	42	53	53

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	68	66	96	91
WPA2-AES	68	66	96	91
WPA3-SAE	68	66	96	91

Mobile AP mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	51	45	57	61
WPA2-AES	42	45	55	56
WPA3-SAE	43	45	57	56

Mobile AP mode throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	83	68	118	101
WPA2-AES	77	68	118	100
WPA3-SAE	77	69	118	100

Parent topic: Wi-Fi throughput

Parent topic: *IW416 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)

- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic: *IW416 release notes*

Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p64.1 to 16.91.21.p82

Component	Description
Wi-Fi	WPA3-R3 enabled APUT beacons does not have RSNXE when configured in H2E mode

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p82 to 16.91.21.p91.6

Component	Description
Wi-Fi	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p91.6 to 16.91.21.p124

Component	Description
Wi-Fi	Cloud keep alive packets not seen after DUT enters host sleep. DUT is sending QOS null packets even in host sleep

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p124 to 16.91.21.p133

Component	Description
Wi-Fi	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p133 to 16.91.21.p133.2

Component	Description
Wi-Fi	DUT STA getting rebooted after 15~20 iterations of 11R-Command based roaming0xa4 command timeout after several hours of stress test

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p133.2 to 16.91.21.p142.5

Component	Description
Wi-Fi	DUT fails to reconnect after the configured auto-reconnect time interval.
Coex	During HFP call, TX side noise is observed with coex CLI

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p142.5 to 16.91.21.p149.4

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p149.4 to 16.92.21.p151.7

Component	Description
Wi-Fi	Samsung S24 Ultra and Google Pixel 7 mobiles having Android 14 are not able connect to the DUTAP with WPA3 SAE security.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p151.7 to 16.92.21.p153.5

Component	Description
Wi-Fi	The DUT encounters a command response timeout during the execution of the wlan-info command following UDP traffic tests.
Wi-Fi	Random hang issue seen when using wlan-p2p-find/stop in succession

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.5 to 16.92.21.p153.6

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.6 to 16.92.21.p153.9

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.9 to 16.92.21.p155.2

Component	Description
Wi-Fi	Enabled mbedtls 3.xWi-Fi connection in 2.4GHz is not stable, observed deauthentication within 10sec.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p155.2 to 16.92.21.p161

Component	Description
Wi-Fi	Fixed STA unable to connect to AP in noisy conditions with antenna diversity enabled.
Bluetooth	Intermittent failure in LE connection establishment with reason code 0x3E

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p161 to 16.92.21.p166

Component	Description
Wi-Fi	Fixed an issue where DHCP renewal was not triggered when roaming between independent APs sharing the same SSID. Fixed an issue where the DUT failed to fall back to legacy roaming after 802.11k/11v roaming returned no usable target AP

Parent topic: Bug fixes and/or feature enhancements

Parent topic: *IW416 release notes*

Known issues

Component	Description
NA	

Parent topic: *IW416 release notes*

IW611/IW612 release notes

Package information

- SDK version: 26.09.00

Parent topic: *IW611/IW612 release notes*

Version information

- Wireless SoC: IW611/IW612
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.8.p141
  - 18 - Major revision

  - 99 - Feature pack
  - 8 - Release version
  - p141 - Patch number

Parent topic: *IW611/IW612 release notes*

Host platform

- All i.MX RT platforms running FreeRTOS except i.MX RT1060 EVKA and i.MX RT1064 EVK
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
  - 802.15.4 over SPI (IW612 only)
- Test tools
  - iPerf (version 2.1.9)

Parent topic: *IW611/IW612 release notes*

Wi-Fi and Bluetooth certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA2
- STA | WPA3
- STA | 802.11ac
- STA | 802.11ax
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification

QDID: refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic: *IW611/IW612 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: IW612 Murata (Module: 2EL M.2) with EVK-MIMXRT1060 EVKC platform
- DUT Power Source: External power supply
- Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*

The throughput numbers are captured with default configurations using *wifi_wpa_supplicant* sample application.

Parent topic: Wi-Fi throughput

iPerf host configuration and impact on throughput

To get the highest throughput, the throughput values shown in STA throughput and Mobile AP throughput are measured with the maximum values of the default host configuration macros. STA and AP throughput captured with the minimum values of the host configuration macros shows the throughput numbers obtained when using the minimum values of the host configuration macros. The macro values are defined in *lwipopts.h* file.

The table below lists the minimum and maximum values of the host configuration macros.

Values of the host configuration macros

Parameter	Maximum value	Minimum value
TCPIP_MBOX_SIZE	96	32
DEFAULT_RAW_RECVMBOX_SIZE	32	12
DEFAULT_UDP_RECVMBOX_SIZE	64	12
DEFAULT_TCP_RECVMBOX_SIZE	64	12
TCP_MSS	1460	536
TCP_SND_BUF	24 * TCP_MSS	2 * TCP_MSS
MEM_SIZE	319160	41,080
TCP_WND	15 * TCP_MSS	10 * TCP_MSS
MEMP_NUM_PBUF	20	10
MEMP_NUM_TCP_SEG	96	12
MEMP_NUM_TCPIP_MSG_INPKT	80	16
MEMP_NUM_TCPIP_MSG_API	80	8
MEMP_NUM_NETBUF	32	16

STA and AP throughput captured with the minimum values of the host configuration macros

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	7	18	111	124
WPA2-AES	7	18	110	124
WPA3-SAE	6	18	110	124

Mobile AP mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	2	19	93	127
WPA2-AES	2	19	105	126
WPA3-SAE	2	19	104	132

Parent topic: iPerf host configuration and impact on throughput

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	46	46	53	62
WPA2-AES	54	55	70	72
WPA3-SAE	53	55	70	72

STA mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	77	77	130	131
WPA2-AES	80	79	130	129
WPA3-SAE	80	79	130	129

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	41	48	63	64
WPA2-AES	56	58	71	75
WPA3-SAE	56	58	71	75

STA mode throughput - AN Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	78	81	129	134
WPA2-AES	78	80	129	133
WPA3-SAE	77	80	129	133

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	45	54	71	76
WPA2-AES	56	58	71	75
WPA3-SAE	56	58	71	75

STA mode throughput - VHT Mode | 5 GHz Band | 40 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	92	86	153	157
WPA2-AES	89	91	153	157
WPA3-SAE	90	87	153	156

STA mode throughput - VHT Mode | 5 GHz Band | 80 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	120	93	233	226
WPA2-AES	114	104	233	226
WPA3-SAE	113	104	233	226

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	69	53	95	102
WPA2-AES	63	65	93	95
WPA3-SAE	66	68	101	99

STA mode throughput - HE Mode | 2.4 GHz Band | 40 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	97	88	214	210
WPA2-AES	99	88	204	209
WPA3-SAE	95	86	216	210

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	59	51	93	127
WPA2-AES	60	51	91	126
WPA3-SAE	60	51	91	127

STA mode throughput - HE Mode | 5 GHz Band | 40 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	103	90	185	228
WPA2-AES	102	91	180	228
WPA3-SAE	101	88	180	228

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	123	97	233	226
WPA2-AES	123	70	233	225
WPA3-SAE	126	70	233	225

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple MacBook Air

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	44	45	58	57
WPA2-AES	43	44	60	56
WPA3-SAE	43	44	59	56

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	69	44	122	114
WPA2-AES	68	43	120	113
WPA3-SAE	68	43	120	112

Mobile AP mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	46	49	63	61
WPA2-AES	46	49	63	60
WPA3-SAE	46	49	63	60

Mobile AP mode throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	75	78	127	113
WPA2-AES	74	78	127	112
WPA3-SAE	75	78	127	112

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	50	52	71	71
WPA2-AES	49	51	71	71
WPA3-SAE	50	51	71	71

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	90	90	152	141
WPA2-AES	89	89	148	139
WPA3-SAE	89	89	144	140

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	116	110	224	214
WPA2-AES	116	110	219	214
WPA3-SAE	115	110	219	214

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	54	48	82	89
WPA2-AES	52	62	81	113
WPA3-SAE	51	62	81	112

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	83	76	133	180
WPA2-AES	62	74	132	132
WPA3-SAE	62	72	130	133

Mobile AP mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	60	52	81	89
WPA2-AES	60	51	80	90
WPA3-SAE	60	51	80	89

Mobile AP mode throughput - HE Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	98	96	150	209
WPA2-AES	97	95	147	214
WPA3-SAE	97	95	147	214

Mobile AP mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	116	107	222	216
WPA2-AES	115	106	217	216
WPA3-SAE	113	106	217	217

Parent topic: Wi-Fi throughput

Parent topic: *IW611/IW612 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)
- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic: *IW611/IW612 release notes*

Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p7.19

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p7.19 to 18.99.2.p49.9

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p49.9 to 18.99.2.p155

Component	Description
Bluetooth	Audio lost occurs due to periodic adv sync lost, during 2 BIS 44.1kHz unencrypted streams with 1M PHY configuration.BIS sync loss may occur in long audio streaming sessions.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p155 to 18.99.2.p66.30

Component	Description
Wi-Fi	802.11R Fast BSS roaming works only with hostapd and does not work with standard APs (supporting 11R)
Bluetooth	DUT is not able to sustain a connection with the remote device that does extended advertisement with coded PHY configuration. When 2 CIS streams are active, after the first device disconnects followed by the second device disconnecting, the second peripheral device hangs.Audio Play/Pause does not work in BIS case.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p66.30 to 18.99.3.p10.5

Component	Description
Wi-Fi	STAUT not sending Neighbor Advertisement packet after receiving Neighbor Solicitation packet from Ex-AP.Antenna selection time exceeds configured evaluation time
Bluetooth	When DUT works as CIS source and CIS Offset is 612us, high packet drops observed which affects the audio streaming.For BIS Source Use Cases, Periodic Interval and ISO Interval should be multiple of each other value.In 1-CIS and 2-CIS, Continuous Audio Glitches are observed with 96 kbps bit rate.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p10.5 to 18.99.3.p17.9

Component	Description
Wi-Fi	After performing independent reset (out-of-band mode), the STAUT fails to connect to the external AP via wlan-connect command, observed command timeout 0x107 error.
Bluetooth	Audio glitches observed with Google Pixel 7 Pro streaming audio after CIS is established with DUT.During Call Gateway (CG) / Call Terminal (CT) Use Case, the firmware periodically sends NULL PDU, which results in frequent Audio Glitch on both CG and CT sides.Heavy audio glitches observed with CIS SRC Google Pixel 7 ProContinuous audio glitches observed in 1 CIS and 2 CIS for 48_3 and 48_4 config.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p17.9 to 18.99.3.p21.154

Component	Description
Wi-Fi	STAUT fail to ping AP backend machine when connected with DFS channel and DUTSTA went in bad state.
Bluetooth	CIS Sink frequently fails to acknowledge CIS Source TX PDU.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p21.154 to 18.99.3.p23.16

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p23.16 to 18.99.3.p25.11

Component	Description
Bluetooth	Packet lost observed in CIS case, which causes audio noise.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p25.11 to 18.99.3.p26.10

Component	Description
Wi-Fi	During legacy roaming when the “Link Lost” observed the DUTSTA fails to roam
Wi-Fi	During the automated testing of the channel performance, a system hang can occur, with the error message “.sdio_drv_write failed”.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p26.10 to 18.99.3.p27.1

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p27.10 to 18.99.8.p10

Component	Description
Wi-Fi	Fail to set band to 2.4G in wifi_test_mode application Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p10 to 18.99.8.p68

Component	Description
Wi-Fi	Duplicate Packet Reception in JP Region due to Missing BA/ACK TransmissionFixed STA setting wrong primary channel when external AP has incorrect secondary channel offset field.Fixed WPA3 STA datapath ping stuck until next reassociation; fixed DUT TX throughput dropping to 0 or getting stuck in pre-sleep state under stress.Fixed scan results taking more than 5 seconds to appear in AP+STA mode.Fixed RSSI_INFO (0xa4) command timeout when DUT is left idle for extended periods in AP+STA mode.
Bluetootl	Sequential Removal of CIS Handles as per current Controller implementation i.e CIS Disconnection sequence should be in sequence => CIS - 4,3,2,1While 4-CIS streaming,audio glitches observed on all CIS SINK with Samsung Galaxy budsFixed missing Disconnect Complete event from controller causing host to get stuck after ACL link disconnection.Added mechanism to detect BT slave link FST on odd-numbered offsetFixed firmware crash observed after eSCO connection complete event.Fixed BTC ACL link disconnection due to CCM counter not being incremented during encryptionFixed BLE and BTC power backoff not working when BTU power control is disabled in MFG firmware.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p68 to 18.99.8.p141

Component	Description
Wi-Fi	The device connects successfully but fails to roam when signal degrades.Fixed an issue where DHCP renewal was not triggered when roaming between independent APs sharing the same SSID Fixed an issue where ping responses failed on the first uAP BSS start in 2.4 GHz Fixed an issue where the DUT failed to fall back to legacy roaming after 802.11k/11v roaming returned no usable target AP
Bluetooth	Fixed an issue where the BR/EDR Coex shell app incorrectly initialized the Nighthawk (IW610) module and downloaded firmware instead of IW612, causing NB UART firmware download to fail with “HDR SIG not found” (-22) and subsequently WLAN initialization to fail.Fixed a sporadic LE RX degradation issue , where GATT indication delays exceeding 500 ms were observed due to excessive retransmissions in a concurrent BTC ACL + eSCO + BLE scatternet scenario. Fixed a issue where DUT sends HCI Hardware Error (0x22)Fixed a issue where DUT establishes LE connections with two devices using same connection handleFixed a issue where Controller will choose connection interval that won’t collide with eSCO link paremeters.

Parent topic: Bug fixes and/or feature enhancements

Parent topic: *IW611/IW612 release notes*

Known issues

Component	Description
NA	

Parent topic: *IW611/IW612 release notes*

AW611 release notes

Package information

- SDK version: 26.09.00

Parent topic: *IW611/IW612 release notes*

Version information

- Wireless SoC: AW611
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.8.p141
  - 18 - Major revision
  - 99 - Feature pack
  - 8 - Release version
  - p141 - Patch number

Parent topic: *AW611 release notes*

Host platform

- i.MX RT1170 EVK, i.MX RT1180 EVK and i.MX RT1180 EVKA Platforms running FreeRTOS
- Host interfaces

  - Wi-Fi over SDIO (SDIO 2.0 support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
- Test tools
  - iPerf (version 2.1.9)

Parent topic: *AW611 release notes*

Wi-Fi and Bluetooth certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA2
- STA | WPA3
- STA | 802.11ac
- STA | 802.11ax
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification

QDID: refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic: *IW611/IW612 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: IW612 Murata (Module: 2EL M.2) with EVK-MIMXRT1060 EVKC platform
- DUT Power Source: External power supply
- Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version gcc-arm-none-eabi-13.2

- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*

The throughput numbers are captured with default configurations using *wifi_wpa_supplicant* sample application.

Parent topic: Wi-Fi throughput

iPerf host configuration and impact on throughput

To get the highest throughput, the throughput values shown in STA throughput and Mobile AP throughput are measured with the maximum values of the default host configuration macros. STA and AP throughput captured with the minimum values of the host configuration macros shows the throughput numbers obtained when using the minimum values of the host configuration macros. The macro values are defined in *lwipopts.h* file.

The table below lists the minimum and maximum values of the host configuration macros.

Values of the host configuration macros

Parameter	Maximum value	Minimum value
TCPIP_MBOX_SIZE	96	32
DEFAULT_RAW_RECVMBOX_SIZE	32	12
DEFAULT_UDP_RECVMBOX_SIZE	64	12
DEFAULT_TCP_RECVMBOX_SIZE	64	12
TCP_MSS	1460	536
TCP_SND_BUF	24 * TCP_MSS	2 * TCP_MSS
MEM_SIZE	319160	41,080
TCP_WND	15 * TCP_MSS	10 * TCP_MSS
MEMP_NUM_PBUF	20	10
MEMP_NUM_TCP_SEG	96	12
MEMP_NUM_TCPIP_MSG_INPKT	80	16
MEMP_NUM_TCPIP_MSG_API	80	8
MEMP_NUM_NETBUF	32	16

STA and AP throughput captured with the minimum values of the host configuration macros

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	7	18	111	124
WPA2-AES	7	18	110	124
WPA3-SAE	6	18	110	124

Mobile AP mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	2	19	93	127
WPA2-AES	2	19	105	126
WPA3-SAE	2	19	104	132

Parent topic: iPerf host configuration and impact on throughput

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	46	46	53	62
WPA2-AES	54	55	70	72
WPA3-SAE	53	55	70	72

STA mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	77	77	130	131
WPA2-AES	80	79	130	129
WPA3-SAE	80	79	130	129

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	41	48	63	64
WPA2-AES	56	58	71	75
WPA3-SAE	56	58	71	75

STA mode throughput - AN Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	78	81	129	134
WPA2-AES	78	80	129	133
WPA3-SAE	77	80	129	133

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	45	54	71	76
WPA2-AES	56	58	71	75
WPA3-SAE	56	58	71	75

STA mode throughput - VHT Mode | 5 GHz Band | 40 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	92	86	153	157
WPA2-AES	89	91	153	157
WPA3-SAE	90	87	153	156

STA mode throughput - VHT Mode | 5 GHz Band | 80 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	120	93	233	226
WPA2-AES	114	104	233	226
WPA3-SAE	113	104	233	226

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	69	53	95	102
WPA2-AES	63	65	93	95
WPA3-SAE	66	68	101	99

STA mode throughput - HE Mode | 2.4 GHz Band | 40 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	97	88	214	210
WPA2-AES	99	88	204	209
WPA3-SAE	95	86	216	210

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	59	51	93	127
WPA2-AES	60	51	91	126
WPA3-SAE	60	51	91	127

STA mode throughput - HE Mode | 5 GHz Band | 40 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	103	90	185	228
WPA2-AES	102	91	180	228
WPA3-SAE	101	88	180	228

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	123	97	233	226
WPA2-AES	123	70	233	225
WPA3-SAE	126	70	233	225

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple MacBook Air

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	44	45	58	57
WPA2-AES	43	44	60	56
WPA3-SAE	43	44	59	56

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	69	44	122	114
WPA2-AES	68	43	120	113
WPA3-SAE	68	43	120	112

Mobile AP mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	46	49	63	61
WPA2-AES	46	49	63	60
WPA3-SAE	46	49	63	60

Mobile AP mode throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	75	78	127	113
WPA2-AES	74	78	127	112
WPA3-SAE	75	78	127	112

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	50	52	71	71
WPA2-AES	49	51	71	71
WPA3-SAE	50	51	71	71

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	90	90	152	141
WPA2-AES	89	89	148	139
WPA3-SAE	89	89	144	140

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	116	110	224	214
WPA2-AES	116	110	219	214
WPA3-SAE	115	110	219	214

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	54	48	82	89
WPA2-AES	52	62	81	113
WPA3-SAE	51	62	81	112

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	83	76	133	180
WPA2-AES	62	74	132	132
WPA3-SAE	62	72	130	133

Mobile AP mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	60	52	81	89
WPA2-AES	60	51	80	90
WPA3-SAE	60	51	80	89

Mobile AP mode throughput - HE Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	98	96	150	209
WPA2-AES	97	95	147	214
WPA3-SAE	97	95	147	214

Mobile AP mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	116	107	222	216
WPA2-AES	115	106	217	216
WPA3-SAE	113	106	217	217

Parent topic: Wi-Fi throughput

Parent topic: *IW611/IW612 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)
- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic: *IW611/IW612 release notes*

Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p7.19

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p7.19 to 18.99.2.p49.9

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p49.9 to 18.99.2.p155

Component	Description
Bluetooth	Audio lost occurs due to periodic adv sync lost, during 2 BIS 44.1kHz unencrypted streams with 1M PHY configuration.BIS sync loss may occur in long audio streaming sessions.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p155 to 18.99.2.p66.30

Component	Description
Wi-Fi	802.11R Fast BSS roaming works only with hostapd and does not work with standard APs (supporting 11R)
Bluetooth	DUT is not able to sustain a connection with the remote device that does extended advertisement with coded PHY configuration. When 2 CIS streams are active, after the first device disconnects followed by the second device disconnecting, the second peripheral device hangs.Audio Play/Pause does not work in BIS case.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p66.30 to 18.99.3.p10.5

Component	Description
Wi-Fi	STAUT not sending Neighbor Advertisement packet after receiving Neighbor Solicitation packet from Ex-AP.Antenna selection time exceeds configured evaluation time
Bluetooth	When DUT works as CIS source and CIS Offset is 612us, high packet drops observed which affects the audio streaming.For BIS Source Use Cases, Periodic Interval and ISO Interval should be multiple of each other value.In 1-CIS and 2-CIS, Continuous Audio Glitches are observed with 96 kbps bit rate.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p10.5 to 18.99.3.p17.9

Component	Description
Wi-Fi	After performing independent reset (out-of-band mode), the STAUT fails to connect to the external AP via wlan-connect command, observed command timeout 0x107 error.
Bluetooth	Audio glitches observed with Google Pixel 7 Pro streaming audio after CIS is established with DUT.During Call Gateway (CG) / Call Terminal (CT) Use Case, the firmware periodically sends NULL PDU, which results in frequent Audio Glitch on both CG and CT sides.Heavy audio glitches observed with CIS SRC Google Pixel 7 ProContinuous audio glitches observed in 1 CIS and 2 CIS for 48_3 and 48_4 config.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p17.9 to 18.99.3.p21.154

Component	Description
Wi-Fi	STAUT fail to ping AP backend machine when connected with DFS channel and DUTSTA went in bad state.
Bluetooth	CIS Sink frequently fails to acknowledge CIS Source TX PDU.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p21.154 to 18.99.3.p23.16

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p23.16 to 18.99.3.p25.11

Component	Description
Bluetooth	Packet lost observed in CIS case, which causes audio noise.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p25.11 to 18.99.3.p26.10

Component	Description
Wi-Fi	During legacy roaming when the “Link Lost” observed the DUTSTA fails to roam
Wi-Fi	During the automated testing of the channel performance, a system hang can occur, with the error message “.sdio_drv_write failed”.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p26.10 to 18.99.3.p27.1

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p27.10 to 18.99.8.p10

Component	Description
Wi-Fi	Fail to set band to 2.4G in wifi_test_mode application Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p10 to 18.99.8.p68

Component	Description
Wi-Fi	Duplicate Packet Reception in JP Region due to Missing BA/ACK TransmissionFixed STA setting wrong primary channel when external AP has incorrect secondary channel offset field.Fixed WPA3 STA datapath ping stuck until next reassociation; fixed DUT TX throughput dropping to 0 or getting stuck in pre-sleep state under stress.Fixed scan results taking more than 5 seconds to appear in AP+STA mode.Fixed RSSI_INFO (0xa4) command timeout when DUT is left idle for extended periods in AP+STA mode.
Bluetootl	Sequential Removal of CIS Handles as per current Controller implementation i.e CIS Disconnection sequence should be in sequence => CIS - 4,3,2,1While 4-CIS streaming,audio glitches observed on all CIS SINK with Samsung Galaxy budsFixed missing Disconnect Complete event from controller causing host to get stuck after ACL link disconnection.Added mechanism to detect BT slave link FST on odd-numbered offsetFixed firmware crash observed after eSCO connection complete event.Fixed BTC ACL link disconnection due to CCM counter not being incremented during encryptionFixed BLE and BTC power backoff not working when BTU power control is disabled in MFG firmware.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p68 to 18.99.8.p141

Component	Description
Wi-Fi	The device connects successfully but fails to roam when signal degrades.Fixed an issue where DHCP renewal was not triggered when roaming between independent APs sharing the same SSID Fixed an issue where ping responses failed on the first uAP BSS start in 2.4 GHz Fixed an issue where the DUT failed to fall back to legacy roaming after 802.11k/11v roaming returned no usable target AP
Bluetootl	Fixed an issue where the BR/EDR Coex shell app incorrectly initialized the Nighthawk (IW610) module and downloaded firmware instead of IW612, causing NB UART firmware download to fail with “HDR SIG not found” (-22) and subsequently WLAN initialization to fail.Fixed a sporadic LE RX degradation issue , where GATT indication delays exceeding 500 ms were observed due to excessive retransmissions in a concurrent BTC ACL + eSCO + BLE scatternet scenario. Fixed a issue where DUT sends HCI Hardware Error (0x22)Fixed a issue where DUT establishes LE connections with two devices using same connection handleFixed a issue where Controller will choose connection interval that won’t collide with eSCO link paremeters.

Parent topic: Bug fixes and/or feature enhancements

Parent topic: *IW611/IW612 release notes*

Known issues

Component	Description
NA	

Parent topic: *IW611/IW612 release notes*

RW610/RW612 release notes

Package information

- SDK version: 26.09.00

Parent topic: *RW610/RW612 release notes*

Version information

- Wi-Fi firmware version: 18.99.8.p141
  - rw61x_sb_wifi_a2.bin for A2
  - 18 - Major revision
  - 99 - Feature pack
  - 8 - Release version
  - p141 - Patch number
- Bluetooth LE firmware version: 18.99.8.p141
  - rw61x_sb_ble_a2.bin for A2
  - 18 - Major revision
  - 99 - Feature pack
  - 8 - Release version
  - p141 - Patch number
- 802.15.4 and Bluetooth LE (up to core 4.1) firmware version: 18.99.8.p141
  - rw61x_sb_ble_15d4_combo_a2.bin for A2
  - 18 - Major revision
  - 99 - Feature pack
  - 8 - Release version
  - p141 - Patch number

Parent topic: *RW610/RW612 release notes*

Host platform

- RW610/RW612 platform running FreeRTOS
- Test tools
  - iPerf (version 2.1.9)

Parent topic: *RW610/RW612 release notes*

Wireless certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD

- STA | WPA2
- STA | WPA3
- STA | 802.11ac
- STA | 802.11ax
- STA | QTT

Refer to *1*.

Note: This release supports STAUT only certifications.

Parent topic: Wireless certification

Bluetooth LE controller certification

QDID: Refer to *4*.

Parent topic: Wireless certification

Thread

Thread group: refer to *7*.

Product Name: NXP RW612 Wireless MCU with Integrated Tri-Radio

Thread version: V1.3.0

CID #: 13A109

Parent topic: Wireless certification

Matter

RW612 certification: refer to *8*.

Certificate ID: CSA23C36MAT41746-24

Device type: Root Node, Thermostat

Transport: Matter over Wi-Fi

RW610 certification: refer to *9*.

Certificate ID: CSA23C43MAT41753-50

Device type: Root Node, Thermostat

Transport: Matter over Wi-Fi and Matter over Thread

Parent topic: Wireless certification

Parent topic: *RW610/RW612 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: RW610/RW612
- External Client: Intel AX210
- Channel: 6 | 36
- Wi-Fi application: wifi_cli

- Compiler used to build application: armgcc
- Compiler version gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *3*.

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	35	39	62	63
WPA2-AES	34	39	60	63
WPA3-SAE	34	39	60	59

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	35	40	64	65
WPA2-AES	35	40	62	64
WPA3-SAE	36	40	62	64

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	38	44	77	77
WPA2-AES	38	43	74	76
WPA3-SAE	38	43	74	76

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	42	45	97	100
WPA2-AES	40	44	96	97
WPA3-SAE	40	44	96	97

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	42	44	100	95
WPA2-AES	41	43	99	92
WPA3-SAE	41	44	99	98

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple MacBook Air

Mobile AP throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	36	39	61	61
WPA2-AES	36	38	59	59
WPA3-SAE	36	38	58	59

Mobile AP throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	38	41	64	64
WPA2-AES	37	40	62	62
WPA3-SAE	37	40	62	62

Mobile AP throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	42	43	75	75
WPA2-AES	41	44	75	74
WPA3-SAE	41	44	75	74

Mobile AP throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	43	47	92	94
WPA2-AES	42	46	93	104
WPA3-SAE	42	46	93	104

Mobile AP throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	45	49	97	98
WPA2-AES	44	48	98	105
WPA3-SAE	44	48	98	105

Parent topic: Wi-Fi throughput

Parent topic: *RW610/RW612 release notes*

Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p34 to 18.99.6.p40

Com-ponent	Description
Zigbee	Zigbee Coordinator and Router are disconnected during BLE connection pairing and bonding with a mobile app for the first time.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p40 to 18.99.6.p46

Compo-nent	Description
Wi-Fi	Fails to establish a persistent connection when the device attempts to reinvoke the second stored Persistent Group
Blue-tooth	NCP cannot work after flash uart bins for both host and device side

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p46 to 18.99.6.p47

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p46 to 18.99.6.p47

Component	Description
Wi-Fi	Enabled mbedtls 3.xHost sleep with WoWLAN fails due to command response timeout (cmd 0x10d) during MCU sleep/wake transitions on WiFi NCP over UART, leading to system assert during sleep exitPing failures observed during concurrent network operations (fast ping from AP backend and DUT), with ICMP unreachable errors indicating routing/forwarding issues and performance degradation under specific network load conditions

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p50 to 18.99.8.p10

Component	Description
Wi-Fi	Enabled mbedtls 3.x Fail to set band to 2.4G in wifi_test_mode application

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p10 to 18.99.8.p68

Component	Description
Wi-Fi	Fixed WPA3 SAE security failure observed in WiFi CLI app during initiating connection with AP. Fixed TX power increase of ~3 dBm observed after connecting to a DFS channel and then reconnecting to a non-DFS channel. TX_POWER_LUT_EXTENSION_SUPPORT did not account for the one-channel DPD case, causing TX power to exceed the configured limit on subsequent non-DFS channel connections.
BLu tootl	Implementation of PaWR (Periodic Advertisement with response) feature
Co-ex-is-tanc	Fixed driver assert issue when wake up wifi cpu1Wi-Fi throughput drops to 0 when Wi-Fi and OT traffic run concurrently on non-overlapping channels at an RSSI of -80 dBm.Fixed high PING latency (up to ~1876 ms average ~397 ms) and packet loss observed with external FEM in Wi-Fi + Thread coexistence mode. RF control lines were switching excessively between Wi-Fi and Narrowband, disrupting Wi-Fi PSM operation and degrading throughput/latency.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p68 to 18.99.8.p141

Component	Description
Wi-Fi	Fixed Wi-Fi RF Test Mode continuous-wave tone transmission stopping when run for a long duration (30 seconds to 15 minutes).Fixed an issue where DHCP renewal was not triggered when roaming between independent APs sharing the same SSID.Fixed an issue where EAP-TLS authentication failed with TLSv1.2 in the wpa_supplicant example.Fixed an issue where wlan-scan-opt parameters incorrectly applied to subsequent normal scans.Fixed an issue where the ip_addr_t type field was left uninitialized in net_configure_dns() when IPv6 was enabled.Fixed an issue where Wi-Fi ceased to function after receiving an 802.11k RRM beacon request for a 5 GHz channel when CONFIG_5GHz_SUPPORT=0.Fixed an issue where the DUT failed to fall back to legacy roaming after 802.11k/11v roaming returned no usable target AP.
Bluetooth LE	Fixed an issue where BLE TX power set above 10 dBm via vendor command 0xFC87 was reset to the OTP/calibration default (~8.3 dBm) upon BLE connection establishment.Fixed an issue where all Bluetooth commands became unresponsive while Wi-Fi continued to function normally after approximately 3–4 hours of Wi-Fi/BT coexistence testing.Fixed an issue where both Wi-Fi and the DUT hung up after approximately 50 iterations of a concurrent Wi-Fi connect/ping/disconnect and BLE init/advertise/disable loop stress test.Fixed an issue where BLE advertising and scanning failed after repeated bt init / bt disable stress cycles (~600 out of 1000 iterations).

Parent topic: Bug fixes and/or feature enhancements

Parent topic: *RW610/RW612 release notes*

Known issues

Component	Description
Wi-Fi	—
Bluetooth LE	—
Zigbee	-
Coex	-

Parent topic: *RW610/RW612 release notes*

IW610 release notes

Package information

- SDK version: 26.09.00

Parent topic: *IW610 release notes*

Version information

- Wireless SoC: IW610
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.8.p141
  - 18 - Major revision
  - 99 - Feature pack
  - 8 - Release version
  - p141 - Patch number

Parent topic: *IW610 release notes*

Host platform

- i.MX RT1170 EVKB, i.MX RT1160 EVK, i.MX RT1060 EVKC, i.MX RT595 EVK, i.MX RT685-AUD EVK, and i.MX RT700 EVK Platforms running FreeRTOS
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
- Test tools
  - iPerf (version 2.1.9)

Parent topic: *IW610 release notes*

Wi-Fi and Bluetooth certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

Bluetooth controller certification

QDID: Refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic: *IW610 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: IW610
- External Client: Intel AX210
- Channel: 6 | 36
- Wi-Fi application: wifi_cli
- Compiler used to build application: armgcc
- Compiler version gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *3*.

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	42	47	60	61
WPA2-AES	41	46	59	59
WPA3-SAE	42	46	58	60

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	46	50	63	64
WPA2-AES	45	49	61	63
WPA3-SAE	45	50	61	62

STA mode throughput - VHT Mode | 2.4 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	41	40	72	72
WPA2-AES	40	40	72	72
WPA3-SAE	40	40	72	71

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	51	58	75	76
WPA2-AES	50	57	73	75
WPA3-SAE	50	57	73	75

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	56	60	93	98
WPA2-AES	53	57	91	91
WPA3-SAE	54	54	94	91

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	50	64	90	102
WPA2-AES	48	61	91	93
WPA3-SAE	48	62	90	93

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple MacBook Air

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	47	44	63	61
WPA2-AES	46	43	61	60
WPA3-SAE	46	43	61	60

Mobile AP mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	49	46	63	63
WPA2-AES	48	45	61	61
WPA3-SAE	48	45	61	61

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	55	51	74	74
WPA2-AES	54	50	74	72
WPA3-SAE	54	50	74	73

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	57	58	95	89
WPA2-AES	55	56	96	84
WPA3-SAE	55	56	96	84

Mobile AP mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	61	58	97	96
WPA2-AES	60	57	97	87
WPA3-SAE	60	57	97	88

Parent topic: Wi-Fi throughput

Parent topic: *IW610 release notes*

Bug fixes and/or feature enhancements

Firmware version: 18.99.5.p66 to 18.99.5.p76

Component	Description
Wi-Fi	The P2P client connection fails when an attempt is made to connect after the P2P Group Owner (P2P-GO) has been stopped.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.5.p76 to 18.99.5.p79

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.5.p79 to 18.99.5.p86

Component	Description
Wi-Fi	Enabled mbedtls 3.xWi-Fi P2P client-GO connection failure due to channel coordination issues between STA and P2P operations; core problem lies in P2P behavior within general Wi-Fi functionality, not RTOS platform-specific features

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.5.p86 to 18.99.8.p10

Component	Description
Wi-Fi	Enabled mbedtls 3.x Fail to set band to 2.4G in wifi_test_mode application

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p10 to 18.99.8.p68

Component	Description
Wi-Fi	Fixed Wi-Fi IOCTL CMD 0x256 response error.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.8.p68 to 18.99.8.p141

Component	Description
Wi-Fi	Fixed Wi-Fi RF Test Mode continuous-wave tone transmission stopping when run for a long duration (30 seconds to 15 minutes).Fixed an issue where DHCP renewal was not triggered when roaming between independent APs sharing the same SSID.Fixed an issue where Wi-Fi initialization failed after a JLink reset, requiring a power cycle to recover.Fixed an issue where the MCU RTOS hung permanently after executing wlan-reset.Fixed an issue where the DUT failed to fall back to legacy roaming after 802.11k/11v roaming returned no usable target AP.
Bluetooth LE	Fixed an issue where the DUT hung during BLE LE Scan when Wi-Fi was associated with a WPA2 security SSID concurrently.Fixed an issue where BLE scan failed to stop cleanly, causing an assertion/hang after issuing bt scan off.
Coex	Fixed an issue where adding a WPA3-SAE or Enterprise Wi-Fi profile caused a crash/hang, preventing Wi-Fi association during concurrent BLE L2CAP coexistence testing.Fixed an issue where BLE re-initialization failed during a BLE init/disable loop test while Wi-Fi was associated and running PING traffic.

Parent topic: Bug fixes and/or feature enhancements

Parent topic: *IW610 release notes*

Known issues

Component	Description
NA	

Parent topic: *IW610 release notes*

Abbreviations

Abbreviation	Definition
A2DP	Advanced audio distribution profile
AMPDU	Aggregated MAC protocol data unit
AMSDU	Aggregated MAC service data unit
AP	Access point
BW	Bandwidth
CCMP	Counter mode CBC-MAC protocol
CSI	Channel state information
CTS	Clear To Send
DL	Down link
EDCA	Enhanced distributed channel access
ER	Extended range

continues on next page

Table 55 – continued from previous page

Abbreviation	Definition
ERP	Extended rate physical
GATT	Generic attribute profile
HFP	Hands free profile
HID	Human interface device
HT	High throughput
LDPC	Low density parity check
MCS	Modulation and coding scheme
MLME	Mac layer management entity
OMI	Operating mode indication
PMF	Protected management frames
RTS	Request to send
SAE	Simultaneous authentication of equals
STA	Station
TWT	Target wake time
UL	Up link
VHT	Very high throughput
WEP	Wired equivalent private
WFD	Wi-Fi direct
WMM	Wireless multi-media
WPA	Wi-Fi protected access
WPS	Wi-Fi protected setup
WSC	Wi-Fi Simple Configuration

References

1. Application note - AN13681 – Wi-Fi Alliance (WFA) Derivative Certification Process (available in the SDK package)
2. User manual – UM11442 - NXP Wi-Fi and Bluetooth Demo Applications User Guide for i.MX RT Platforms (available in the SDK package)
3. User manual – UM11799 - NXP Wi-Fi and Bluetooth Demo Applications User Guide for RW61x (available in the SDK package)
4. Certification – Bluetooth controller - QDID (*link*)
5. User manual - UM12133 - NXP NCP Application Guide for RW612 with MCU Host
6. Technical note - TN00066 – Wi-Fi Alliance (WFA) Derivative Certification Process (available in the SDK package)
7. Web page – Thread certified products (link)
8. Web page – Connectivity standard alliance (csa) – NXP RW612 Tri-Radio Wireless MCU Development Platform (link)
9. Web page – Connectivity standard alliance (csa) – NXP RW610 Wireless MCU Development Platform (link)
10. Application note - AN14634 – Kconfig Memory Optimizer (link)

Chapter 4

RTOS

4.1 FreeRTOS

4.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

FreeRTOS kernel for MCUXpresso SDK Readme

MCUXpresso SDK: FreeRTOS Kernel NXP Integration This repository is the NXP integration layer for the FreeRTOS kernel (https://github.com/FreeRTOS/FreeRTOS-Kernel). It allows original FreeRTOS kernel upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

For more information about the FreeRTOS kernel repo adoption see README_mcuxsdk.md: FreeRTOS kernel for MCUXpresso SDK Readme document.

FreeRTOS kernel for MCUXpresso SDK Overview

The purpose of this document is to describes the FreeRTOS kernel repo integration into the NXP MCUXpresso Software Development Kit: mcuxsdk. MCUXpresso SDK provides a comprehensive development solutions designed to optimize, ease, and help accelerate embedded system development of applications based on MCUs from NXP. This project involves the FreeRTOS kernel repo fork with:

- cmake and Kconfig support to allow the configuration and build in MCUXpresso SDK ecosystem
- FreeRTOS OS additions, such as FreeRTOS driver wrappers, RTOS ready FatFs file system, and the implementation of FreeRTOS tickless mode

The history of changes in FreeRTOS kernel repo for MCUXpresso SDK are summarized in *CHANGELOG.md* file.

The MCUXpresso SDK framework also contains a set of FreeRTOS examples which show basic FreeRTOS OS features. This makes it easy to start a new FreeRTOS project or begin experimenting with FreeRTOS OS. Selected drivers and middleware are RTOS ready with related FreeRTOS adaptation layer.

FreeRTOS example applications

The FreeRTOS examples are written to demonstrate basic FreeRTOS features and the interaction between peripheral drivers and the RTOS.

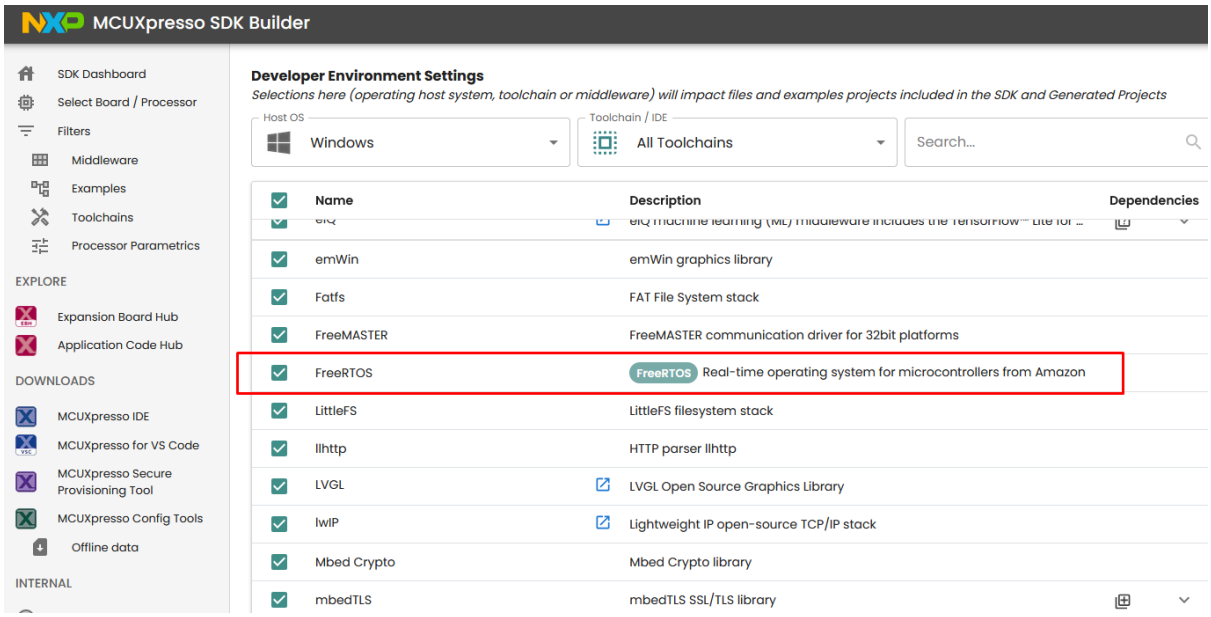
List of examples

The list of freertos_examples, their description and availability for individual supported MCUXpresso SDK development boards can be obtained here: https://mcuxpresso.nxp.com/mcuxsdk/latest/html/examples/freertos_examples/index.html

Location of examples

The FreeRTOS examples are located in mcuxsdk-examples repository, see the freertos_examples folder.

Once using MCUXpresso SDK zip packages created via the MCUXpresso SDK Builder the FreeRTOS kernel library and associated freertos_examples are added into final zip package once FreeRTOS components is selected on the Developer Environment Settings page:



The FreeRTOS examples in MCUXpresso SDK zip packages are located in <MCUXpressoSDK_install_dir>/boards/<board_name>/freertos_examples/ subfolders.

Building a FreeRTOS example application

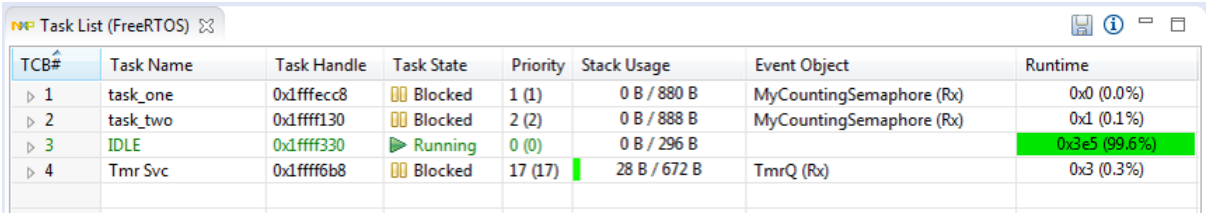
For information how to use the cmake and Kconfig based build and configuration system and how to build freertos_examples visit: MCUXpresso SDK documentation for Build And Configuration MCUXpresso SDK Getting Start Guide

Tip: To list all FreeRTOS example projects and targets that can be built via the west build command, use this west list_project command in mcuxsdk workspace:

```
west list_project -p examples/freertos_examples
```

FreeRTOS aware debugger plugin

NXP provides FreeRTOS task aware debugger for GDB. The plugin is compatible with Eclipse-based (MCUXpressoIDE) and is available after the installation.



TCB#	Task Name	Task Handle	Task State	Priority	Stack Usage	Event Object	Runtime
1	task_one	0x1fffecc8	Blocked	1 (1)	0 B / 880 B	MyCountingSemaphore (Rx)	0x0 (0.0%)
2	task_two	0x1ffff130	Blocked	2 (2)	0 B / 888 B	MyCountingSemaphore (Rx)	0x1 (0.1%)
3	IDLE	0x1ffff330	Running	0 (0)	0 B / 296 B		0x3e5 (99.6%)
4	Tmr Svc	0x1ffff6b8	Blocked	17 (17)	28 B / 672 B	TmrQ (Rx)	0x3 (0.3%)

FreeRTOS kernel for MCUXpresso SDK ChangeLog

Changelog FreeRTOS kernel for MCUXpresso SDK All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

[Unreleased]

Fixed

- Temporary workaround for FreeRTOS-Kernel upstream bug #1480: stray URL (http://files.iar.com/ftp/pub/box/bxarm-9.60.3.deb) accidentally placed outside /* */ comment delimiters on line 466 of the IAR CM33/CM55/CM85 TrustZone non-secure portasm.s files (introduced in upstream commit 78e0cc778a, PR #1195). Fixed copies are placed under portable/IAR/ARM_CM{33,55,85}/non_secure/portasm.s and the CMakeLists.txt SOURCES entries for the cm33/cm55/cm85_trustzone.non_secure IAR toolchain blocks are redirected to those copies. Revert the three CMakeLists.txt lines once the upstream fix is merged and the freertos-kernel-upstream revision is bumped past it, then delete portable/IAR/ARM_CM{33,55,85}/ from this repo.

Added

[11.2.0_rev0]

- update amazon freertos version

[11.1.0_rev0]

- update amazon freertos version

[11.0.1_rev0]

- update amazon freertos version

[10.5.1_rev0]

- update amazon freertos version

[10.4.3_rev1]

- Apply CM33 security fix from 10.4.3-LTS-Patch-2. See rtos\freertos\freertos_kernel\History.txt
- Apply CM33 security fix from 10.4.3-LTS-Patch-1. See rtos\freertos\freertos_kernel\History.txt

[10.4.3_rev0]

- update amazon freertos version.

[10.4.3_rev0]

- update amazon freertos version.

[9.0.0_rev3]

- New features:
  - Tickless idle mode support for Cortex-A7. Add fsl_tickless_epit.c and fsl_tickless_generic.h in portable/IAR/ARM_CA9 folder.
  - Enabled float context saving in IAR for Cortex-A7. Added configUSE_TASK_FPU_SUPPORT macros. Modified port.c and portmacro.h in portable/IAR/ARM_CA9 folder.
- Other changes:
  - Transformed ARM_CM core specific tickless low power support into generic form under freertos/Source/portable/low_power_tickless/.

[9.0.0_rev2]

- New features:
  - Enabled MCUXpresso thread aware debugging. Add freertos_tasks_c_additions.h and configINCLUDE_FREERTOS_TASK_C_ADDITIONS_H and configFREERTOS_MEMORY_SCHEME macros.

[9.0.0_rev1]

- New features:
  - Enabled -flto optimization in GCC by adding **attribute**((used)) for vTaskSwitchContext.
  - Enabled KDS Task Aware Debugger. Apply FreeRTOS patch to enable configRECORD_STACK_HIGH_ADDRESS macro. Modified files are task.c and FreeRTOS.h.

[9.0.0_rev0]

- New features:
  - Example freertos_sem_static.
  - Static allocation support RTOS driver wrappers.
- Other changes:
  - Tickless idle rework. Support for different timers is in separated files (fsl_tickless_systick.c, fsl_tickless_lptmr.c).
  - Removed configuration option configSYSTICK_USE_LOW_POWER_TIMER. Low power timer is now selected by linking of apropriate file fsl_tickless_lptmr.c.
  - Removed configOVERRIDE_DEFAULT_TICK_CONFIGURATION in RVDS port. Use of **attribute**((weak)) is the preferred solution. Not same as _weak!

[8.2.3]

- New features:
  - Tickless idle mode support.
  - Added template application for Kinetis Expert (KEx) tool (template_application).
- Other changes:
  - Folder structure reduction. Keep only Kinetis related parts.

FreeRTOS kernel Readme

4.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

4.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

Readme

MCUXpresso SDK: backoffAlgorithm Library NXP Integration This repository is the NXP integration layer for the backoffAlgorithm library (https://github.com/FreeRTOS/backoffalgorithm). It allows original backoffAlgorithm upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.4 corehttp

C language HTTP client library designed for embedded platforms.

MCUXpresso SDK: coreHTTP Client Library NXP Integration

This repository is the NXP integration layer for the coreHTTP Client library (https://github.com/FreeRTOS/corehttp). It allows original coreHTTP Client upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.5 corejson

JSON parser.

Readme

MCUXpresso SDK: coreJSON Library NXP Integration This repository is the NXP integration layer for the coreJSON library (https://github.com/FreeRTOS/corejson). It allows original coreJSON upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.6 coremqtt

MQTT publish/subscribe messaging library.

MCUXpresso SDK: coreMQTT Library NXP Integration

This repository is the NXP integration layer for the coreMQTT library (https://github.com/FreeRTOS/coremqtt). It allows original coreMQTT upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.7 corepkcs11

PKCS #11 key management library.

Readme

MCUXpresso SDK: corePKCS11 Library NXP Integration This repository is the NXP integration layer for the PKCS #11 key management library (https://github.com/FreeRTOS/corePKCS11). It allows original PKCS #11 key management upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

4.1.8 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

Readme

MCUXpresso SDK: FreeRTOS-Plus-TCP Library NXP Integration This repository is the NXP integration layer for the FreeRTOS-Plus-TCP library (https://github.com/FreeRTOS/freertos-plus-tcp). It allows original FreeRTOS-Plus-TCP upstream sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Chapter 5

Downloads

This page is generated automatically by the “downloads_index“ Sphinx extension.
4 downloadable file(s) referenced across 3 page(s), grouped into 2 categories:

5.1 boards

- mcuxsdk-mimxrt700evk.pdf — from *boards/RT/mimxrt700evk/index*

5.2 develop

- evkbmimxrt1170_custom (.zip) — from develop/sdk/custom_board_development
- frdmmcxa346_custom (.zip) — from develop/sdk/custom_board_development
- example (.zip) — from develop/sdk/integration

Bibliography

[Security-Incident-Process] Security Incident Handling Process

[FF-M] Arm® Platform Security Architecture Firmware Framework 1.0

[FF-M-1.1-Extensions] Arm® Firmware Framework for M 1.1 Extensions

[DUAL-CPU-BOOT] *Booting a dual core system*

[CVSS] Common Vulnerability Scoring System Version 3.1 Calculator

[CVSS_SPEC] CVSS v3.1 Specification Document

[STRIDE] The STRIDE Threat Model

[SECURE-BOOT] *Secure boot*

[ROLLBACK-PROTECT] *Rollback protection in TF-M secure boot*

[Arm-ARM] Armv8-M Architecture Reference Manual

[STACK-SEAL] Armv8-M processor Secure software Stack Sealing vulnerability

[ADVISORY-TFMV-1] *Advisory TFMV-1*

[ADVISORY-TFMV-2] *Advisory TFMV-2*

[VLLDM-Vulnerability] VLLDM instruction Security Vulnerability

[PSA-FF-M] Arm Platform Security Architecture Firmware Framework 1.0

[RFC7925] Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things

[PROFILE-S] *Trusted Firmware-M Profile Small Design*

[RFC7252] The Constrained Application Protocol (CoAP)

[RFC4279] Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)

[RFC7251] AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS

[CRYPTO-DESIGN] *Crypto design*

[ITS-INTEGRATE] *ITS integration guide*

[TFM-BUILD] *TF-M build instruction*

[a] PS service is enabled by default. Platforms without off-chip storage devices can turn off TFM_PARTITION_PROTECTED_STORAGE to disable PS service. See *Protected Storage Secure Partition* for details.

[TBSA-M] Arm Platform Security Architecture Trusted Base System Architecture for Armv6-M, Armv7-M and Armv8-M, version 1.0

[HKDF] Hugo Krawczyk. 2010. Cryptographic extraction and key derivation: the HKDF scheme. In Proceedings of the 30th annual conference on Advances in cryptology (CRYPTO'10)

[RFC5869] IETF RFC 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF)

Index

A

AAPCS, **1126**
Application RoT, **1126**

D

drivers_RT_RT700_MIMXRT798S.___cdog_debug_Action_ctrl_enum (*C enum*), 209
drivers_RT_RT700_MIMXRT798S.___cdog_debug_Action_ctrl_enum.kCDOG_DebugHaltCtrl_Pause (*C enumerator*), 209
drivers_RT_RT700_MIMXRT798S.___cdog_debug_Action_ctrl_enum.kCDOG_DebugHaltCtrl_Run (*C enumerator*), 209
drivers_RT_RT700_MIMXRT798S.___cdog_fault_ctrl_enum (*C enum*), 209
drivers_RT_RT700_MIMXRT798S.___cdog_fault_ctrl_enum.kCDOG_FaultCtrl_EnableInterrupt (*C enumerator*), 209
drivers_RT_RT700_MIMXRT798S.___cdog_fault_ctrl_enum.kCDOG_FaultCtrl_EnableReset (*C enumerator*), 209
drivers_RT_RT700_MIMXRT798S.___cdog_fault_ctrl_enum.kCDOG_FaultCtrl_NoAction (*C enumerator*), 209
drivers_RT_RT700_MIMXRT798S.___cdog_irq_pause_ctrl_enum (*C enum*), 209
drivers_RT_RT700_MIMXRT798S.___cdog_irq_pause_ctrl_enum.kCDOG_IrqPauseCtrl_Pause (*C enumerator*), 209
drivers_RT_RT700_MIMXRT798S.___cdog_irq_pause_ctrl_enum.kCDOG_IrqPauseCtrl_Run (*C enumerator*), 209
drivers_RT_RT700_MIMXRT798S.___code_lock_ctrl_enum (*C enum*), 209
drivers_RT_RT700_MIMXRT798S.___code_lock_ctrl_enum.kCDOG_LockCtrl_Lock (*C enumerator*), 210
drivers_RT_RT700_MIMXRT798S.___code_lock_ctrl_enum.kCDOG_LockCtrl_Unlock (*C enumerator*), 210
drivers_RT_RT700_MIMXRT798S.___pad0___ (*C var*), 536
drivers_RT_RT700_MIMXRT798S.___pad1___ (*C var*), 537
drivers_RT_RT700_MIMXRT798S.___pad2___ (*C var*), 537
drivers_RT_RT700_MIMXRT798S.___pad3___ (*C var*), 537
drivers_RT_RT700_MIMXRT798S.___pad4___ (*C var*), 537
drivers_RT_RT700_MIMXRT798S._acmp_channel_config (*C struct*), 201
drivers_RT_RT700_MIMXRT798S._acmp_channel_config.minusMuxInput (*C var*), 201
drivers_RT_RT700_MIMXRT798S._acmp_channel_config.negativePortInput (*C var*), 201
drivers_RT_RT700_MIMXRT798S._acmp_channel_config.plusMuxInput (*C var*), 201
drivers_RT_RT700_MIMXRT798S._acmp_channel_config.positivePortInput (*C var*), 201
drivers_RT_RT700_MIMXRT798S._acmp_config (*C struct*), 200
drivers_RT_RT700_MIMXRT798S._acmp_config.enableHighSpeed (*C var*), 200
drivers_RT_RT700_MIMXRT798S._acmp_config.enableInvertOutput (*C var*), 200
drivers_RT_RT700_MIMXRT798S._acmp_config.enablePinOut (*C var*), 201
drivers_RT_RT700_MIMXRT798S._acmp_config.hysteresisMode (*C var*), 200
drivers_RT_RT700_MIMXRT798S._acmp_config.offsetMode (*C var*), 200
drivers_RT_RT700_MIMXRT798S._acmp_config.useUnfilteredOutput (*C var*), 201
drivers_RT_RT700_MIMXRT798S._acmp_dac_config (*C struct*), 201
drivers_RT_RT700_MIMXRT798S._acmp_dac_config.DACValue (*C var*), 202

drivers_RT_RT700_MIMXRT798S._acmp_dac_config.enableOutput (*C var*), 202
drivers_RT_RT700_MIMXRT798S._acmp_dac_config.referenceVoltageSource (*C var*), 201
drivers_RT_RT700_MIMXRT798S._acmp_dac_work_mode (*C enum*), 199
drivers_RT_RT700_MIMXRT798S._acmp_dac_work_mode.kACMP_DACWorkHighSpeedMode (*C enumerator*), 199
drivers_RT_RT700_MIMXRT798S._acmp_dac_work_mode.kACMP_DACWorkLowSpeedMode (*C enumerator*), 199
drivers_RT_RT700_MIMXRT798S._acmp_discrete_mode_config (*C struct*), 202
drivers_RT_RT700_MIMXRT798S._acmp_discrete_mode_config.enableNegativeChannelDiscreteMode (*C var*), 202
drivers_RT_RT700_MIMXRT798S._acmp_discrete_mode_config.enablePositiveChannelDiscreteMode (*C var*), 202
drivers_RT_RT700_MIMXRT798S._acmp_filter_config (*C struct*), 201
drivers_RT_RT700_MIMXRT798S._acmp_filter_config.filterCount (*C var*), 201
drivers_RT_RT700_MIMXRT798S._acmp_filter_config.filterPeriod (*C var*), 201
drivers_RT_RT700_MIMXRT798S._acmp_hysteresis_mode (*C enum*), 198
drivers_RT_RT700_MIMXRT798S._acmp_hysteresis_mode.kACMP_HysteresisLevel0 (*C enumerator*), 198
drivers_RT_RT700_MIMXRT798S._acmp_hysteresis_mode.kACMP_HysteresisLevel1 (*C enumerator*), 199
drivers_RT_RT700_MIMXRT798S._acmp_hysteresis_mode.kACMP_HysteresisLevel2 (*C enumerator*), 199
drivers_RT_RT700_MIMXRT798S._acmp_hysteresis_mode.kACMP_HysteresisLevel3 (*C enumerator*), 199
drivers_RT_RT700_MIMXRT798S._acmp_interrupt_enable (*C enum*), 198
drivers_RT_RT700_MIMXRT798S._acmp_interrupt_enable.kACMP_OutputFallingInterruptEnable (*C enumerator*), 198
drivers_RT_RT700_MIMXRT798S._acmp_interrupt_enable.kACMP_OutputRisingInterruptEnable (*C enumerator*), 198
drivers_RT_RT700_MIMXRT798S._acmp_offset_mode (*C enum*), 198
drivers_RT_RT700_MIMXRT798S._acmp_offset_mode.kACMP_OffsetLevel0 (*C enumerator*), 198
drivers_RT_RT700_MIMXRT798S._acmp_offset_mode.kACMP_OffsetLevel1 (*C enumerator*), 198
drivers_RT_RT700_MIMXRT798S._acmp_port_input (*C enum*), 199
drivers_RT_RT700_MIMXRT798S._acmp_port_input.kACMP_PortInputFromDAC (*C enumerator*), 199
drivers_RT_RT700_MIMXRT798S._acmp_port_input.kACMP_PortInputFromMux (*C enumerator*), 199
drivers_RT_RT700_MIMXRT798S._acmp_reference_voltage_source (*C enum*), 199
drivers_RT_RT700_MIMXRT798S._acmp_reference_voltage_source.kACMP_VrefSourceVin1 (*C enumerator*), 199
drivers_RT_RT700_MIMXRT798S._acmp_reference_voltage_source.kACMP_VrefSourceVin2 (*C enumerator*), 199
drivers_RT_RT700_MIMXRT798S._acmp_status_flags (*C enum*), 198
drivers_RT_RT700_MIMXRT798S._acmp_status_flags.kACMP_OutputAssertEventFlag (*C enumerator*), 198
drivers_RT_RT700_MIMXRT798S._acmp_status_flags.kACMP_OutputFallingEventFlag (*C enumerator*), 198
drivers_RT_RT700_MIMXRT798S._acmp_status_flags.kACMP_OutputRisingEventFlag (*C enumerator*), 198
drivers_RT_RT700_MIMXRT798S._arm2ezhv_intctl (*C enum*), 326
drivers_RT_RT700_MIMXRT798S._arm2ezhv_intctl.kEZHV_ARM2EZHV_MEI (*C enumerator*), 326
drivers_RT_RT700_MIMXRT798S._arm2ezhv_intctl.kEZHV_ARM2EZHV_MSI (*C enumerator*), 326
drivers_RT_RT700_MIMXRT798S._arm2ezhv_intctl.kEZHV_ARM2EZHV_MTI (*C enumerator*), 326
drivers_RT_RT700_MIMXRT798S._arm2ezhv_intctl.kEZHV_ARM2EZHV_SEI (*C enumerator*), 326

drivers_RT_RT700_MIMXRT798S._audio_pll_mult (*C enum*), 244
drivers_RT_RT700_MIMXRT798S._audio_pll_mult.kCLOCK_AudioPllMult16 (*C enumerator*), 244
drivers_RT_RT700_MIMXRT798S._audio_pll_mult.kCLOCK_AudioPllMult17 (*C enumerator*), 244
drivers_RT_RT700_MIMXRT798S._audio_pll_mult.kCLOCK_AudioPllMult18 (*C enumerator*), 244
drivers_RT_RT700_MIMXRT798S._audio_pll_mult.kCLOCK_AudioPllMult19 (*C enumerator*), 244
drivers_RT_RT700_MIMXRT798S._audio_pll_mult.kCLOCK_AudioPllMult20 (*C enumerator*), 244
drivers_RT_RT700_MIMXRT798S._audio_pll_mult.kCLOCK_AudioPllMult21 (*C enumerator*), 244
drivers_RT_RT700_MIMXRT798S._audio_pll_mult.kCLOCK_AudioPllMult22 (*C enumerator*), 244
drivers_RT_RT700_MIMXRT798S._audio_pll_src (*C enum*), 244
drivers_RT_RT700_MIMXRT798S._audio_pll_src.kCLOCK_AudioPllFro1Div8 (*C enumerator*), 244
drivers_RT_RT700_MIMXRT798S._audio_pll_src.kCLOCK_AudioPllOscClk (*C enumerator*), 244
drivers_RT_RT700_MIMXRT798S._body_bias_domain (*C enum*), 805
drivers_RT_RT700_MIMXRT798S._body_bias_domain.kPower_BodyBiasVdd1 (*C enumerator*), 805
drivers_RT_RT700_MIMXRT798S._body_bias_domain.kPower_BodyBiasVdd1Sram (*C enumerator*), 805
drivers_RT_RT700_MIMXRT798S._body_bias_domain.kPower_BodyBiasVdd2 (*C enumerator*), 805
drivers_RT_RT700_MIMXRT798S._body_bias_domain.kPower_BodyBiasVdd2Sram (*C enumerator*), 805
drivers_RT_RT700_MIMXRT798S._body_bias_domain.kPower_BodyBiasVddn (*C enumerator*), 805
drivers_RT_RT700_MIMXRT798S._cache64_config (*C struct*), 206
drivers_RT_RT700_MIMXRT798S._cache64_config.boundaryAddr (*C var*), 206
drivers_RT_RT700_MIMXRT798S._cache64_policy (*C enum*), 205
drivers_RT_RT700_MIMXRT798S._cache64_policy.kCACHE64_PolicyNonCacheable (*C enumerator*), 205
drivers_RT_RT700_MIMXRT798S._cache64_policy.kCACHE64_PolicyWriteBack (*C enumerator*), 205
drivers_RT_RT700_MIMXRT798S._cache64_policy.kCACHE64_PolicyWriteThrough (*C enumerator*), 205
drivers_RT_RT700_MIMXRT798S._clock_attach_id (*C enum*), 226
drivers_RT_RT700_MIMXRT798S._clock_attach_id.k32KHZ_WAKE_to_CTIMER5 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.k32KHZ_WAKE_to_CTIMER6 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.k32KHZ_WAKE_to_CTIMER7 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.k32KHZ_WAKE_to_EUSB (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.k32KHZ_WAKE_to_FLEXCOMM17 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.k32KHZ_WAKE_to_FLEXCOMM18 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.k32KHZ_WAKE_to_FLEXCOMM19 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.k32KHZ_WAKE_to_FLEXCOMM20 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.k32KHZ_WAKE_to_OSTIMER (*C enumerator*), 232

drivers_RT_RT700_MIMXRT798S._clock_attach_id.k32KHZ_WAKE_to_SYSTICK (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.k32KHZ_WAKE_to_USB (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_PFD0_to_SDIO0 (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_PFD0_to_SDIO1 (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_PFD1_to_LCDIF (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_PFD1_to_SENSE_DSP (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_PFD1_to_XSPI2 (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_PFD2_to_MIPI_DPHYESC_CLK (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_PFD2_to_MIPI_DSI_HOST_PHY (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_PFD2_to_SENSE_RAM (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_PFD3_to_AUDIO_VDD1 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_PFD3_to_MICFIL0 (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_PFD3_to_SENSE_MAIN (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_PFD3_to_VDD1_CLKOUT (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_PLL_to_SDADC (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_VDD1_to_CTIMER5 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_VDD1_to_CTIMER6 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_VDD1_to_CTIMER7 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_VDD1_to_MICFIL0 (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_VDD1_to_SAI3 (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kAUDIO_VDD1_to_SDADC (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kCOMMON_BASE_to_COMMON_VDDN (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kCOMMON_BASE_to_XSPI2 (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kEUSB_24MHZ_to_EUSB (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV1_to_FLEXIO (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV1_to_LCDIF (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV1_to_LPSPI14 (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV1_to_LPSPI16 (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV1_to_MEDIA_MAIN (*C enumerator*), 234

drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV1_to_MEDIA_VDDN (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV1_to_MIPI_DPHYESC_CLK (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV1_to_MIPI_DSI_HOST_PHY (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV1_to_SDIO0 (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV1_to_SDIO1 (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV1_to_VGPU (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV1_to_XSPI2 (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV3_to_COMMON_BASE (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV3_to_MEDIA_VDD2_BASE (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO0_DIV3_to_MEDIA_VDDN_BASE (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_COMMON_BASE (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_COMMON_VDDN (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_FLEXCOMM17 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_FLEXCOMM18 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_FLEXCOMM19 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_FLEXCOMM20 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_FLEXIO (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_LPI2C15 (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_LPSPI14 (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_LPSPI16 (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_MEDIA_VDD2_BASE (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_MEDIA_VDDN_BASE (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_SENSE_BASE (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_SENSE_DSP (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_SENSE_MAIN (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_SENSE_RAM (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV1_to_VDD1_CLKOUT (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV2_to_LPI2C15 (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV2_to_UTICK (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV3_to_COMMON_BASE (*C enu-*

merator), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV3_to_MEDIA_VDD2_BASE (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV3_to_MEDIA_VDDN_BASE (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV3_to_SENSE_BASE (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV8_to_AUDIO_PLL0 (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV8_to_EUSB_24MHZ (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV8_to_I3C23 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV8_to_MAIN_PLL0 (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO1_DIV8_to_USB_24MHZ (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_ADC (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_CTIMER5 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_CTIMER6 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_CTIMER7 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_FLEXCOMM17 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_FLEXCOMM18 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_FLEXCOMM19 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_FLEXCOMM20 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_I3C23 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_LPI2C15 (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_MICFIL0 (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_SAI3 (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_SENSE_DSP (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_SENSE_MAIN (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_SENSE_RAM (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_UTICK (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV1_to_VDD1_CLKOUT (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV3_to_SENSE_BASE (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV8_to_AUDIO_VDD1 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV8_to_SAI3 (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kFRO2_DIV8_to_SDADC (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kLPOSC_DIV32_to_32K_WAKE (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kLPOSC_to_COMMON_BASE (*C enumerator*), 231

drivers_RT_RT700_MIMXRT798S._clock_attach_id.kLPOSC_to_EUSB (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kLPOSC_to_MEDIA_VDD2_BASE (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kLPOSC_to_MEDIA_VDDN_BASE (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kLPOSC_to_OSTIMER (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kLPOSC_to_SENSE_BASE (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kLPOSC_to_SYSTICK (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kLPOSC_to_USB (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kLPOSC_to_UTICK (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kLPOSC_to_WWDT2 (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kLPOSC_to_WWDT3 (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD0_to_LPSPI14 (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD0_to_LPSPI16 (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD0_to_MEDIA_MAIN (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD0_to_MEDIA_VDDN (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD0_to_VGPU (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD1_to_ADC (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD1_to_MIPI_DPHYESC_CLK (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD1_to_SDIO1 (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD2_to_LCDIF (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD2_to_MEDIA_MAIN (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD2_to_MEDIA_VDDN (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD2_to_SDIO0 (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD2_to_VGPU (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD3_to_COMMON_VDDN (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD3_to_FLEXIO (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMAIN_PLL_PFD3_to_XSPI2 (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMCLK_to_AUDIO_VDD1 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMEDIA_VDD2_BASE_to_FLEXIO (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMEDIA_VDD2_BASE_to_LCDIF (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMEDIA_VDD2_BASE_to_LPSPI14 (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMEDIA_VDD2_BASE_to_LPSPI16 (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMEDIA_VDD2_BASE_to_MEDIA_MAIN (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMEDIA_VDD2_BASE_to_MIPI_DPHYESC_CLK

(*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMEDIA_VDD2_BASE_to_MIPI_DSI_HOST_PHY (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMEDIA_VDD2_BASE_to_VGPU (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMEDIA_VDDN_BASE_to_MEDIA_VDDN (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMEDIA_VDDN_BASE_to_SDIO0 (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kMEDIA_VDDN_BASE_to_SDIO1 (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_32K_WAKE (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_ADC (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_CTIMER5 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_CTIMER6 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_CTIMER7 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_EUSB (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_FLEXCOMM17 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_FLEXCOMM18 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_FLEXCOMM19 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_FLEXCOMM20 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_FLEXIO (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_I3C23 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_LCDIF (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_LPI2C15 (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_LPSPI14 (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_LPSPI16 (*C enumerator*), 238
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_MICFIL0 (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_MIPI_DPHYESC_CLK (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_MIPI_DSI_HOST_PHY (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_OSTIMER (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_SAI3 (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_SDADC (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_SDIO0 (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_SDIO1 (*C enumerator*), 236
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_SENSE_DSP (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_SYSTICK (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_USB (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_UTICK (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_VDD1_CLKOUT (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_VGPU (*C enumerator*), 237
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_WWDT2 (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_WWDT3 (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kNONE_to_XSPI2 (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kOSC32K_to_32K_WAKE (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kOSC_CLK_to_ADC (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kOSC_CLK_to_AUDIO_PLL0 (*C enumerator*), 231

drivers_RT_RT700_MIMXRT798S._clock_attach_id.kOSC_CLK_to_COMMON_VDDN (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kOSC_CLK_to_EUSB_24MHZ (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kOSC_CLK_to_I3C23 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kOSC_CLK_to_MAIN_PLL0 (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kOSC_CLK_to_OSTIMER (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kOSC_CLK_to_SYSTICK (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kOSC_CLK_to_USB_24MHZ (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kOSC_to_AUDIO_VDD1 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_ADC (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_CTIMER5 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_CTIMER6 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_CTIMER7 (*C enumerator*), 228
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_FLEXCOMM17 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_FLEXCOMM18 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_FLEXCOMM19 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_FLEXCOMM20 (*C enumerator*), 230
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_I3C23 (*C enumerator*), 229
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_LPI2C15 (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_MICFIL0 (*C enumerator*), 233
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_OSTIMER (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_SAI3 (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_SDADC (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_SENSE_DSP (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_SENSE_MAIN (*C enumerator*), 231
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_SENSE_RAM (*C enumerator*), 232
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_SYSTICK (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_UTICK (*C enumerator*), 227
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kSENSE_BASE_to_VDD1_CLKOUT (*C enumerator*), 234
drivers_RT_RT700_MIMXRT798S._clock_attach_id.kUSB_24MHZ_to_USB (*C enumerator*), 235
drivers_RT_RT700_MIMXRT798S._clock_audio_pll_config (*C struct*), 267
drivers_RT_RT700_MIMXRT798S._clock_div_name (*C enum*), 238

drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivAdcClk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivCommonVddnClk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivCtimer5Clk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivCtimer6Clk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivCtimer7Clk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivDphyClk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivDphyEscRxClk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivDphyEscTxClk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivFlexioClk (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivI3c23Clk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivLcdifClk (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivLowFreqClk (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivLPFlexComm17Clk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivLPFlexComm18Clk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivLPFlexComm19Clk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivLPFlexComm20Clk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivLpi2c15Clk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivLpspi14Clk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivLpspi16Clk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivMediaMainClk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivMediaVddnClk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivMicfil0Clk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivOstimerClk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivSai3Clk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivSdadcClk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivSdio0Clk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivSdio1Clk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivSenseDspClk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivSenseMainClk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivSenseRamClk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivSystickClk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivUtick1Clk (*C enumerator*), 239
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivVdd1ClockOut (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivVgpuClk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_div_name.kCLOCK_DivXspi2Clk (*C enumerator*), 240
drivers_RT_RT700_MIMXRT798S._clock_domain_enable (*C enum*), 242
drivers_RT_RT700_MIMXRT798S._clock_domain_enable.kCLOCK_AllDomainEnable (*C enumerator*), 242
drivers_RT_RT700_MIMXRT798S._clock_domain_enable.kCLOCK_Vdd1SenseDomainEnable (*C enumerator*), 242

drivers_RT_RT700_MIMXRT798S._clock_domain_enable.kCLOCK_Vdd2ComDomainEnable (*C enumerator*), 242
drivers_RT_RT700_MIMXRT798S._clock_domain_enable.kCLOCK_Vdd2CompDomainEnable (*C enumerator*), 242
drivers_RT_RT700_MIMXRT798S._clock_domain_enable.kCLOCK_Vdd2DspDomainEnable (*C enumerator*), 242
drivers_RT_RT700_MIMXRT798S._clock_domain_enable.kCLOCK_Vdd2MediaDomainEnable (*C enumerator*), 242
drivers_RT_RT700_MIMXRT798S._clock_domain_enable.kCLOCK_VddnComDomainEnable (*C enumerator*), 242
drivers_RT_RT700_MIMXRT798S._clock_domain_enable.kCLOCK_VddnMediaDomainEnable (*C enumerator*), 242
drivers_RT_RT700_MIMXRT798S._clock_fro_config (*C struct*), 267
drivers_RT_RT700_MIMXRT798S._clock_fro_flag (*C enum*), 242
drivers_RT_RT700_MIMXRT798S._clock_fro_flag.kCLOCK_FroLossOfLockErr (*C enumerator*), 242
drivers_RT_RT700_MIMXRT798S._clock_fro_flag.kCLOCK_FroTrimLock (*C enumerator*), 242
drivers_RT_RT700_MIMXRT798S._clock_fro_flag.kCLOCK_FroTrimUpdateReq (*C enumerator*), 242
drivers_RT_RT700_MIMXRT798S._clock_fro_flag.kCLOCK_FroTuneErr (*C enumerator*), 242
drivers_RT_RT700_MIMXRT798S._clock_fro_flag.kCLOCK_FroTuneOnceDone (*C enumerator*), 242
drivers_RT_RT700_MIMXRT798S._clock_fro_interrupt (*C enum*), 241
drivers_RT_RT700_MIMXRT798S._clock_fro_interrupt.kCLOCK_FroAllIntterrupt (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_fro_interrupt.kCLOCK_FroLossOfLockInt (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_fro_interrupt.kCLOCK_FroTrimUpdateReqInt (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_fro_interrupt.kCLOCK_FroTuneErrInt (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_fro_output_en (*C enum*), 241
drivers_RT_RT700_MIMXRT798S._clock_fro_output_en.kCLOCK_FroAllOutEn (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_fro_output_en.kCLOCK_FroDiv1OutEn (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_fro_output_en.kCLOCK_FroDiv2OutEn (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_fro_output_en.kCLOCK_FroDiv3OutEn (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_fro_output_en.kCLOCK_FroDiv6OutEn (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_fro_output_en.kCLOCK_FroDiv8OutEn (*C enumerator*), 241
drivers_RT_RT700_MIMXRT798S._clock_ip_name (*C enum*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Acmp0 (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Adc0 (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_AxbsEzh (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Axi0 (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Cache64ctrl0 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Cache64ctrl1 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Cdog0 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Cdog1 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Cdog2 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Cdog3 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Cdog4 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_CompAccessRamArbiter1 (*C enumerator*), 218

drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Cpu1 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Crc0 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Ct32b0 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Ct32b1 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Ct32b2 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Ct32b3 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Ct32b4 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Ct32b5 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Ct32b6 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Ct32b7 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Dbg (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Dma0 (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Dma1 (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Dma2 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Dma3 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Ezhv (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Flexio (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Freqme0 (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gdet3 (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gdet3Ref (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Glikey0 (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Glikey1 (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Glikey2 (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Glikey3 (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Glikey4 (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Glikey5 (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gpio0 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gpio1 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gpio2 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gpio3 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gpio4 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gpio5 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gpio6 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gpio7 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gpio8 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gpio9 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gpio10 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Gpu (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Hifi1 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Hifi4 (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Hifi4AccessRamArbiter1 (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_I3c0 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_I3c1 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_I3c2 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_I3c3 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_InputMux0 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_InputMux1 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Iopctl0 (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Iopctl1 (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Iopctl2 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_IpInvalid (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_JpgDecoder (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Lcdif (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm0 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm1 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm2 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm3 (*C enumerator*), 213

drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm4 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm5 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm6 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm7 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm8 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm9 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm10 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm11 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm12 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm13 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm17 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm18 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm19 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPFlexComm20 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c0 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c1 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c2 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c3 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c4 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c5 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c6 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c7 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c8 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c9 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c10 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c11 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c12 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c13 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c15 (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c17 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c18 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c19 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPI2c20 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi0 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi1 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi2 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi3 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi4 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi5 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi6 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi7 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi8 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi9 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi10 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi11 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi12 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi13 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi14 (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi16 (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi17 (*C enumerator*), 219

drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi18 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi19 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPSpi20 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart0 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart1 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart2 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart3 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart4 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart5 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart6 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart7 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart8 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart9 (*C enumerator*), 214
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart10 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart11 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart12 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart13 (*C enumerator*), 215
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart17 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart18 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart19 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_LPUart20 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_MediaAccessRamArbiter0 (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_MediaAccessRamArbiter1 (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_MipiDsiCtrl (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Mmu0 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Mmu1 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Mmu2 (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Mrt0 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Mrt1 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Mu0 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Mu1 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Mu2 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Mu3 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Mu4 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Npu0 (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Ocotp0 (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_OsTimer (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Pdm (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Pint0 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Pint1 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Pkc (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_PkcRam (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_PngDecoder (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Prince0 (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Prince1 (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_PrinceExe (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Puf (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Pvts0 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Pvts1 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Romcp (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Rtc (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_SafoSgi (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Sai0 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Sai1 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Sai2 (*C enumerator*), 216
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Sai3 (*C enumerator*), 220

drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Sct0 (*C enumerator*), 213
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Sdadc0 (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Sema420 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Sema423 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Sema424 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_SenseAccessRamArbiter0 (*C enumerator*), 219
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Sleepcon0 (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Sleepcon1 (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_SWOTraceClkin (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Syscon0 (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Syscon1 (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Syscon2 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Syscon3 (*C enumerator*), 222
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Syscon4 (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Syspm0 (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Syspm1 (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_TpiuTraceClkin (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Trace (*C enumerator*), 218
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Tsclk (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Usb0 (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Usb1 (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_UsbphyRef (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Usdhc0 (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Usdhc1 (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Utick0 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Utick1 (*C enumerator*), 220
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Wwdt0 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Wwdt1 (*C enumerator*), 217
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Wwdt2 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Wwdt3 (*C enumerator*), 221
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Xcache0 (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Xcache1 (*C enumerator*), 211
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Xspi0 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Xspi1 (*C enumerator*), 212
drivers_RT_RT700_MIMXRT798S._clock_ip_name.kCLOCK_Xspi2 (*C enumerator*), 223
drivers_RT_RT700_MIMXRT798S._clock_main_pll_config (*C struct*), 267
drivers_RT_RT700_MIMXRT798S._clock_name (*C enum*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_AcmpClk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_AdcClk (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_BusClk (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_CoreSysClk (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_FlexioClk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_Hifi1CpuClk (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_I3cClk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_LcdifClk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_LPFlexComm17Clk (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_LPFlexComm18Clk (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_LPFlexComm19Clk (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_LPFlexComm20Clk (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_LPI2c15Clk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_LPSpi14Clk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_LPSpi16Clk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_MclkClk (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_MipiDphyClk (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_MipiDphyEscRxClk (*C enumerator*), 226

drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_MipiDphyEscTxClk (*C enumerator*), 226
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_PdmClk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_SctClk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_Sdio0Clk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_Sdio1Clk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_SystickClk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_Usb0Clk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_Usb1Clk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_Vdd1ClockOutClk (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_VgpuClk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_Wdt2Clk (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_Wdt3Clk (*C enumerator*), 224
drivers_RT_RT700_MIMXRT798S._clock_name.kCLOCK_Xspi2Clk (*C enumerator*), 225
drivers_RT_RT700_MIMXRT798S._clock_osc32k_config_t (*C struct*), 267
drivers_RT_RT700_MIMXRT798S._clock_pfd (*C enum*), 243
drivers_RT_RT700_MIMXRT798S._clock_pfd.kCLOCK_Pfd0 (*C enumerator*), 243
drivers_RT_RT700_MIMXRT798S._clock_pfd.kCLOCK_Pfd1 (*C enumerator*), 243
drivers_RT_RT700_MIMXRT798S._clock_pfd.kCLOCK_Pfd2 (*C enumerator*), 243
drivers_RT_RT700_MIMXRT798S._clock_pfd.kCLOCK_Pfd3 (*C enumerator*), 243
drivers_RT_RT700_MIMXRT798S._clock_usb_phy_src (*C enum*), 210
drivers_RT_RT700_MIMXRT798S._clock_usb_phy_src.kCLOCK_Usbphy480M (*C enumerator*), 210
drivers_RT_RT700_MIMXRT798S._clock_usb_src (*C enum*), 210
drivers_RT_RT700_MIMXRT798S._clock_usb_src.kCLOCK_Usb480M (*C enumerator*), 210
drivers_RT_RT700_MIMXRT798S._clock_usb_src.kCLOCK_UsbSrcUnused (*C enumerator*), 210
drivers_RT_RT700_MIMXRT798S._crc_bits (*C enum*), 268
drivers_RT_RT700_MIMXRT798S._crc_bits.kCrcBits16 (*C enumerator*), 268
drivers_RT_RT700_MIMXRT798S._crc_bits.kCrcBits32 (*C enumerator*), 268
drivers_RT_RT700_MIMXRT798S._crc_config (*C struct*), 270
drivers_RT_RT700_MIMXRT798S._crc_config.complementChecksum (*C var*), 270
drivers_RT_RT700_MIMXRT798S._crc_config.crcBits (*C var*), 270
drivers_RT_RT700_MIMXRT798S._crc_config.crcResult (*C var*), 270
drivers_RT_RT700_MIMXRT798S._crc_config.polynomial (*C var*), 270
drivers_RT_RT700_MIMXRT798S._crc_config.reflectIn (*C var*), 270
drivers_RT_RT700_MIMXRT798S._crc_config.reflectOut (*C var*), 270
drivers_RT_RT700_MIMXRT798S._crc_config.seed (*C var*), 270
drivers_RT_RT700_MIMXRT798S._crc_result (*C enum*), 268
drivers_RT_RT700_MIMXRT798S._crc_result.kCrcFinalChecksum (*C enumerator*), 268
drivers_RT_RT700_MIMXRT798S._crc_result.kCrcIntermediateChecksum (*C enumerator*), 268
drivers_RT_RT700_MIMXRT798S._ctimer_capture_channel (*C enum*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_capture_channel.kCTIMER_Capture_0 (*C enumerator*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_capture_channel.kCTIMER_Capture_1 (*C enumerator*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_capture_channel.kCTIMER_Capture_3 (*C enumerator*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_capture_edge (*C enum*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_capture_edge.kCTIMER_Capture_BothEdge (*C enumerator*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_capture_edge.kCTIMER_Capture_FallEdge (*C enumerator*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_capture_edge.kCTIMER_Capture_RiseEdge (*C enumerator*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_config (*C struct*), 281
drivers_RT_RT700_MIMXRT798S._ctimer_config.input (*C var*), 281
drivers_RT_RT700_MIMXRT798S._ctimer_config.mode (*C var*), 281
drivers_RT_RT700_MIMXRT798S._ctimer_config.prescale (*C var*), 281
drivers_RT_RT700_MIMXRT798S._ctimer_external_match (*C enum*), 275

drivers_RT_RT700_MIMXRT798S._ctimer_external_match.kCTIMER_External_Match_0 (*C enumerator*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_external_match.kCTIMER_External_Match_1 (*C enumerator*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_external_match.kCTIMER_External_Match_2 (*C enumerator*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_external_match.kCTIMER_External_Match_3 (*C enumerator*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_interrupt_enable (*C enum*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_interrupt_enable.kCTIMER_Match0InterruptEnable (*C enumerator*), 276
drivers_RT_RT700_MIMXRT798S._ctimer_interrupt_enable.kCTIMER_Match1InterruptEnable (*C enumerator*), 276
drivers_RT_RT700_MIMXRT798S._ctimer_interrupt_enable.kCTIMER_Match2InterruptEnable (*C enumerator*), 276
drivers_RT_RT700_MIMXRT798S._ctimer_interrupt_enable.kCTIMER_Match3InterruptEnable (*C enumerator*), 276
drivers_RT_RT700_MIMXRT798S._ctimer_match (*C enum*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_match_config (*C struct*), 280
drivers_RT_RT700_MIMXRT798S._ctimer_match_config.enableCounterReset (*C var*), 280
drivers_RT_RT700_MIMXRT798S._ctimer_match_config.enableCounterStop (*C var*), 280
drivers_RT_RT700_MIMXRT798S._ctimer_match_config.enableInterrupt (*C var*), 281
drivers_RT_RT700_MIMXRT798S._ctimer_match_config.matchValue (*C var*), 280
drivers_RT_RT700_MIMXRT798S._ctimer_match_config.outControl (*C var*), 280
drivers_RT_RT700_MIMXRT798S._ctimer_match_config.outPinInitState (*C var*), 280
drivers_RT_RT700_MIMXRT798S._ctimer_match_output_control (*C enum*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_match_output_control.kCTIMER_Output_Clear (*C enumerator*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_match_output_control.kCTIMER_Output_NoAction (*C enumerator*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_match_output_control.kCTIMER_Output_Set (*C enumerator*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_match_output_control.kCTIMER_Output_Toggle (*C enumerator*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_match.kCTIMER_Match_0 (*C enumerator*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_match.kCTIMER_Match_1 (*C enumerator*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_match.kCTIMER_Match_2 (*C enumerator*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_match.kCTIMER_Match_3 (*C enumerator*), 274
drivers_RT_RT700_MIMXRT798S._ctimer_status_flags (*C enum*), 276
drivers_RT_RT700_MIMXRT798S._ctimer_status_flags.kCTIMER_Match0Flag (*C enumerator*), 276
drivers_RT_RT700_MIMXRT798S._ctimer_status_flags.kCTIMER_Match1Flag (*C enumerator*), 276
drivers_RT_RT700_MIMXRT798S._ctimer_status_flags.kCTIMER_Match2Flag (*C enumerator*), 276
drivers_RT_RT700_MIMXRT798S._ctimer_status_flags.kCTIMER_Match3Flag (*C enumerator*), 276
drivers_RT_RT700_MIMXRT798S._ctimer_timer_mode (*C enum*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_timer_mode.kCTIMER_IncreaseOnBothEdge (*C enumerator*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_timer_mode.kCTIMER_IncreaseOnFallEdge (*C enumerator*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_timer_mode.kCTIMER_IncreaseOnRiseEdge (*C enumerator*), 275
drivers_RT_RT700_MIMXRT798S._ctimer_timer_mode.kCTIMER_TimerMode (*C enumerator*), 275
drivers_RT_RT700_MIMXRT798S._dsi_config (*C struct*), 718
drivers_RT_RT700_MIMXRT798S._dsi_config.autoInsertEoTp (*C var*), 718

drivers_RT_RT700_MIMXRT798S._dsi_config.btaTo_ByteClk (*C var*), 718
drivers_RT_RT700_MIMXRT798S._dsi_config.enableNonContinuousHsClk (*C var*), 718
drivers_RT_RT700_MIMXRT798S._dsi_config.enableTxUlps (*C var*), 718
drivers_RT_RT700_MIMXRT798S._dsi_config.htxTo_ByteClk (*C var*), 718
drivers_RT_RT700_MIMXRT798S._dsi_config.lrxHostTo_ByteClk (*C var*), 718
drivers_RT_RT700_MIMXRT798S._dsi_config.numExtraEoTp (*C var*), 718
drivers_RT_RT700_MIMXRT798S._dsi_config.numLanes (*C var*), 718
drivers_RT_RT700_MIMXRT798S._dsi_dbi_pixel_format (*C enum*), 707
drivers_RT_RT700_MIMXRT798S._dsi_dbi_pixel_format.kDSI_DbiNoMapping (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S._dsi_dbi_pixel_format.kDSI_DbiRGB332 (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S._dsi_dbi_pixel_format.kDSI_DbiRGB444 (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S._dsi_dbi_pixel_format.kDSI_DbiRGB565 (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S._dsi_dbi_pixel_format.kDSI_DbiRGB666 (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S._dsi_dbi_pixel_format.kDSI_DbiRGB888 (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S._dsi_dphy_config (*C struct*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dphy_config.tClkPost_ByteClk (*C var*), 720
drivers_RT_RT700_MIMXRT798S._dsi_dphy_config.tClkPre_ByteClk (*C var*), 720
drivers_RT_RT700_MIMXRT798S._dsi_dphy_config.tClkPrepare_HalfEscClk (*C var*), 720
drivers_RT_RT700_MIMXRT798S._dsi_dphy_config.tClkTrail_ByteClk (*C var*), 720
drivers_RT_RT700_MIMXRT798S._dsi_dphy_config.tClkZero_ByteClk (*C var*), 720
drivers_RT_RT700_MIMXRT798S._dsi_dphy_config.tHsExit_ByteClk (*C var*), 720
drivers_RT_RT700_MIMXRT798S._dsi_dphy_config.tHsPrepare_HalfEscClk (*C var*), 720
drivers_RT_RT700_MIMXRT798S._dsi_dphy_config.tHsTrail_ByteClk (*C var*), 720
drivers_RT_RT700_MIMXRT798S._dsi_dphy_config.tHsZero_ByteClk (*C var*), 720
drivers_RT_RT700_MIMXRT798S._dsi_dphy_config.tWakeup_EscClk (*C var*), 720
drivers_RT_RT700_MIMXRT798S._dsi_dphy_config.txHsBitClk_Hz (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_bllp_mode (*C enum*), 709
drivers_RT_RT700_MIMXRT798S._dsi_dpi_bllp_mode.kDSI_DpiBllpBlanking (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S._dsi_dpi_bllp_mode.kDSI_DpiBllpLowPower (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S._dsi_dpi_bllp_mode.kDSI_DpiBllpNull (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S._dsi_dpi_color_coding (*C enum*), 707
drivers_RT_RT700_MIMXRT798S._dsi_dpi_color_coding.kDSI_Dpi16BitConfig1 (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S._dsi_dpi_color_coding.kDSI_Dpi16BitConfig2 (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S._dsi_dpi_color_coding.kDSI_Dpi16BitConfig3 (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S._dsi_dpi_color_coding.kDSI_Dpi18BitConfig1 (*C enumerator*), 708
drivers_RT_RT700_MIMXRT798S._dsi_dpi_color_coding.kDSI_Dpi18BitConfig2 (*C enumerator*), 708
drivers_RT_RT700_MIMXRT798S._dsi_dpi_color_coding.kDSI_Dpi24Bit (*C enumerator*), 708
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config (*C struct*), 718
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.bllpMode (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.dpiColorCoding (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.hbp (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.hfp (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.hsw (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.panelHeight (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.pixelPacket (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.pixelPayloadSize (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.polarityFlags (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.vbp (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.vfp (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.videoMode (*C var*), 719

drivers_RT_RT700_MIMXRT798S._dsi_dpi_config.virtualChannel (*C var*), 719
drivers_RT_RT700_MIMXRT798S._dsi_dpi_pixel_packet (*C enum*), 708
drivers_RT_RT700_MIMXRT798S._dsi_dpi_pixel_packet.kDSI_PixelPacket16Bit (*C enumerator*), 708
drivers_RT_RT700_MIMXRT798S._dsi_dpi_pixel_packet.kDSI_PixelPacket18Bit (*C enumerator*), 708
drivers_RT_RT700_MIMXRT798S._dsi_dpi_pixel_packet.kDSI_PixelPacket18BitLoosely (*C enumerator*), 708
drivers_RT_RT700_MIMXRT798S._dsi_dpi_pixel_packet.kDSI_PixelPacket24Bit (*C enumerator*), 708
drivers_RT_RT700_MIMXRT798S._dsi_dpi_video_mode (*C enum*), 708
drivers_RT_RT700_MIMXRT798S._dsi_dpi_video_mode.kDSI_DpiBurst (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S._dsi_dpi_video_mode.kDSI_DpiNonBurstWithSyncEvent (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S._dsi_dpi_video_mode.kDSI_DpiNonBurstWithSyncPulse (*C enumerator*), 708
drivers_RT_RT700_MIMXRT798S._dsi_handle (*C struct*), 721
drivers_RT_RT700_MIMXRT798S._dsi_handle.callback (*C var*), 721
drivers_RT_RT700_MIMXRT798S._dsi_handle.isBusy (*C var*), 721
drivers_RT_RT700_MIMXRT798S._dsi_handle.userData (*C var*), 721
drivers_RT_RT700_MIMXRT798S._dsi_handle.xfer (*C var*), 721
drivers_RT_RT700_MIMXRT798S._dsi_host_status (*C enum*), 710
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostAckTriggerReceived (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostChecksumError (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostContentionDetected (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostEccErrorMultiBit (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostEccErrorOneBit (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostEoTSyncError (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostEscEntryCmdError (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostFalseControlError (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostInvalidDataType (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostInvalidTxLength (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostInvalidVcId (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostLpTxSyncError (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostPeriphToError (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostProtocalViolation (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostResetTriggerReceived (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostSoTError (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostSoTSyncError (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S._dsi_host_status.kDSI_HostTearTriggerReceived (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S._dsi_rx_data_type (*C enum*), 716
drivers_RT_RT700_MIMXRT798S._dsi_rx_data_type.kDSI_RxDataAckAndErrorReport (*C enumerator*), 716
drivers_RT_RT700_MIMXRT798S._dsi_rx_data_type.kDSI_RxDataDcsLongRdResponse (*C enumerator*), 716

drivers_RT_RT700_MIMXRT798S._dsi_rx_data_type.kDSI_RxDataDcsShortRdResponseOneByte (*C enumerator*), 716
drivers_RT_RT700_MIMXRT798S._dsi_rx_data_type.kDSI_RxDataDcsShortRdResponseTwoByte (*C enumerator*), 716
drivers_RT_RT700_MIMXRT798S._dsi_rx_data_type.kDSI_RxDataEoTp (*C enumerator*), 716
drivers_RT_RT700_MIMXRT798S._dsi_rx_data_type.kDSI_RxDataGenLongRdResponse (*C enumerator*), 716
drivers_RT_RT700_MIMXRT798S._dsi_rx_data_type.kDSI_RxDataGenShortRdResponseOneByte (*C enumerator*), 716
drivers_RT_RT700_MIMXRT798S._dsi_rx_data_type.kDSI_RxDataGenShortRdResponseTwoByte (*C enumerator*), 716
drivers_RT_RT700_MIMXRT798S._dsi_transfer (*C struct*), 720
drivers_RT_RT700_MIMXRT798S._dsi_transfer.dcsCmd (*C var*), 721
drivers_RT_RT700_MIMXRT798S._dsi_transfer.flags (*C var*), 721
drivers_RT_RT700_MIMXRT798S._dsi_transfer.rxData (*C var*), 721
drivers_RT_RT700_MIMXRT798S._dsi_transfer.rxDataSize (*C var*), 721
drivers_RT_RT700_MIMXRT798S._dsi_transfer.sendDcsCmd (*C var*), 721
drivers_RT_RT700_MIMXRT798S._dsi_transfer.txData (*C var*), 721
drivers_RT_RT700_MIMXRT798S._dsi_transfer.txDataSize (*C var*), 721
drivers_RT_RT700_MIMXRT798S._dsi_transfer.txDataType (*C var*), 721
drivers_RT_RT700_MIMXRT798S._dsi_transfer.virtualChannel (*C var*), 720
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type (*C enum*), 714
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataBlanking (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataCmOff (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataCmOn (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataDcsLongWr (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataDcsShortRdNoParam (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataDcsShortWrNoParam (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataDcsShortWrOneParam (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataEoTp (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataGenLongWr (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataGenShortRdNoParam (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataGenShortRdOneParam (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataGenShortRdTwoParam (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataGenShortWrNoParam (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataGenShortWrOneParam (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataGenShortWrTwoParam (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataHsyncEnd (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataHsyncStart (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataLooselyPackedPixel18BitRGB (*C enumerator*), 716
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataLooselyPackedPixel20BitYCbCr (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataNull (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataPackedPixel12BitYCrCb (*C*

enumerator), 716
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataPackedPixel16BitRGB (*C enumerator*), 716
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataPackedPixel16BitYCbCr (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataPackedPixel18BitRGB (*C enumerator*), 716
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataPackedPixel24BitRGB (*C enumerator*), 716
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataPackedPixel24BitYCbCr (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataPackedPixel30BitRGB (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataPackedPixel36BitRGB (*C enumerator*), 716
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataSetMaxReturnPktSize (*C enumerator*), 715
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataShutDownPeriph (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataTurnOnPeriph (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataVsyncEnd (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S._dsi_tx_data_type.kDSI_TxDataVsyncStart (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S._dsp_copy_image (*C struct*), 282
drivers_RT_RT700_MIMXRT798S._dsp_static_vec_sel (*C enum*), 281
drivers_RT_RT700_MIMXRT798S._dsp_static_vec_sel.kDSP_StatVecSel0 (*C enumerator*), 281
drivers_RT_RT700_MIMXRT798S._dsp_static_vec_sel.kDSP_StatVecSel1 (*C enumerator*), 281
drivers_RT_RT700_MIMXRT798S._edma4_core_tcd (*C struct*), 323
drivers_RT_RT700_MIMXRT798S._edma5_core_tcd (*C struct*), 322
drivers_RT_RT700_MIMXRT798S._edma5_core_tcd.RESERVED (*C var*), 323
drivers_RT_RT700_MIMXRT798S._edma_bandwidth (*C enum*), 304
drivers_RT_RT700_MIMXRT798S._edma_bandwidth.kEDMA_BandwidthStall4Cycle (*C enumerator*), 305
drivers_RT_RT700_MIMXRT798S._edma_bandwidth.kEDMA_BandwidthStall8Cycle (*C enumerator*), 305
drivers_RT_RT700_MIMXRT798S._edma_bandwidth.kEDMA_BandwidthStallNone (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_channel_access_type (*C enum*), 308
drivers_RT_RT700_MIMXRT798S._edma_channel_access_type.kEDMA_ChannelDataAccess (*C enumerator*), 308
drivers_RT_RT700_MIMXRT798S._edma_channel_access_type.kEDMA_ChannelInstructionAccess (*C enumerator*), 309
drivers_RT_RT700_MIMXRT798S._edma_channel_config (*C struct*), 312
drivers_RT_RT700_MIMXRT798S._edma_channel_config.channelAccessType (*C var*), 312
drivers_RT_RT700_MIMXRT798S._edma_channel_config.channelDataSignExtensionBitPosition (*C var*), 312
drivers_RT_RT700_MIMXRT798S._edma_channel_config.channelPreemptionConfig (*C var*), 312
drivers_RT_RT700_MIMXRT798S._edma_channel_config.channelReadMemoryAttribute (*C var*), 312
drivers_RT_RT700_MIMXRT798S._edma_channel_config.channelRequestSource (*C var*), 312
drivers_RT_RT700_MIMXRT798S._edma_channel_config.channelSwapSize (*C var*), 312
drivers_RT_RT700_MIMXRT798S._edma_channel_config.channelWriteMemoryAttribute (*C var*), 312
drivers_RT_RT700_MIMXRT798S._edma_channel_config.enableMasterIDReplication (*C var*), 312
drivers_RT_RT700_MIMXRT798S._edma_channel_config.protectionLevel (*C var*), 312
drivers_RT_RT700_MIMXRT798S._edma_channel_link_type (*C enum*), 305

drivers_RT_RT700_MIMXRT798S._edma_channel_link_type.kEDMA_LinkNone (*C enumerator*), 305
drivers_RT_RT700_MIMXRT798S._edma_channel_link_type.kEDMA_MajorLink (*C enumerator*), 305
drivers_RT_RT700_MIMXRT798S._edma_channel_link_type.kEDMA_MinorLink (*C enumerator*), 305
drivers_RT_RT700_MIMXRT798S._edma_channel_Preemption_config (*C struct*), 311
drivers_RT_RT700_MIMXRT798S._edma_channel_Preemption_config.channelPriority (*C var*), 311
drivers_RT_RT700_MIMXRT798S._edma_channel_Preemption_config.enableChannelPreemption (*C var*), 311
drivers_RT_RT700_MIMXRT798S._edma_channel_Preemption_config.enablePreemptAbility (*C var*), 311
drivers_RT_RT700_MIMXRT798S._edma_channel_protection_level (*C enum*), 309
drivers_RT_RT700_MIMXRT798S._edma_channel_protection_level.kEDMA_ChannelProtectionLevelPrivileged (*C enumerator*), 309
drivers_RT_RT700_MIMXRT798S._edma_channel_protection_level.kEDMA_ChannelProtectionLevelUser (*C enumerator*), 309
drivers_RT_RT700_MIMXRT798S._edma_channel_swap_size (*C enum*), 308
drivers_RT_RT700_MIMXRT798S._edma_channel_swap_size.kEDMA_ChannelReadWith8bitSwap (*C enumerator*), 308
drivers_RT_RT700_MIMXRT798S._edma_channel_swap_size.kEDMA_ChannelReadWith16bitSwap (*C enumerator*), 308
drivers_RT_RT700_MIMXRT798S._edma_channel_swap_size.kEDMA_ChannelReadWith32bitSwap (*C enumerator*), 308
drivers_RT_RT700_MIMXRT798S._edma_channel_swap_size.kEDMA_ChannelSwapDisabled (*C enumerator*), 308
drivers_RT_RT700_MIMXRT798S._edma_channel_swap_size.kEDMA_ChannelWriteWith8bitSwap (*C enumerator*), 308
drivers_RT_RT700_MIMXRT798S._edma_channel_swap_size.kEDMA_ChannelWriteWith16bitSwap (*C enumerator*), 308
drivers_RT_RT700_MIMXRT798S._edma_channel_swap_size.kEDMA_ChannelWriteWith32bitSwap (*C enumerator*), 308
drivers_RT_RT700_MIMXRT798S._edma_config (*C struct*), 315
drivers_RT_RT700_MIMXRT798S._edma_config.channelConfig (*C var*), 315
drivers_RT_RT700_MIMXRT798S._edma_config.enableDebugMode (*C var*), 315
drivers_RT_RT700_MIMXRT798S._edma_config.enableGlobalChannelLink (*C var*), 315
drivers_RT_RT700_MIMXRT798S._edma_config.enableHaltOnError (*C var*), 315
drivers_RT_RT700_MIMXRT798S._edma_config.enableMasterIdReplication (*C var*), 315
drivers_RT_RT700_MIMXRT798S._edma_config.enableRoundRobinArbitration (*C var*), 315
drivers_RT_RT700_MIMXRT798S._edma_core_channel (*C struct*), 321
drivers_RT_RT700_MIMXRT798S._edma_core_channel.CH_REGS (*C union*), 325
drivers_RT_RT700_MIMXRT798S._edma_core_channel.CH_REGS.EDMA4_REG (*C struct*), 325
drivers_RT_RT700_MIMXRT798S._edma_core_channel.CH_REGS.EDMA4_REG (*C var*), 325
drivers_RT_RT700_MIMXRT798S._edma_core_channel.CH_REGS.EDMA5_REG (*C struct*), 325
drivers_RT_RT700_MIMXRT798S._edma_core_channel.CH_REGS.EDMA5_REG (*C var*), 325
drivers_RT_RT700_MIMXRT798S._edma_core_mp (*C struct*), 321
drivers_RT_RT700_MIMXRT798S._edma_core_mp.MP_REGS (*C union*), 324
drivers_RT_RT700_MIMXRT798S._edma_core_mp.MP_REGS.EDMA5_REG (*C struct*), 324
drivers_RT_RT700_MIMXRT798S._edma_core_mp.MP_REGS.EDMA5_REG (*C var*), 324
drivers_RT_RT700_MIMXRT798S._edma_core_tcd (*C struct*), 324
drivers_RT_RT700_MIMXRT798S._edma_core_tcd.TCD_REGS (*C union*), 325
drivers_RT_RT700_MIMXRT798S._edma_core_tcd.TCD_REGS.edma4_tcd (*C var*), 325
drivers_RT_RT700_MIMXRT798S._edma_handle (*C struct*), 315
drivers_RT_RT700_MIMXRT798S._edma_handle.base (*C var*), 316
drivers_RT_RT700_MIMXRT798S._edma_handle.callback (*C var*), 315
drivers_RT_RT700_MIMXRT798S._edma_handle.channel (*C var*), 316
drivers_RT_RT700_MIMXRT798S._edma_handle.channelBase (*C var*), 316
drivers_RT_RT700_MIMXRT798S._edma_handle.header (*C var*), 316

drivers_RT_RT700_MIMXRT798S._edma_handle.tail (*C var*), 316
drivers_RT_RT700_MIMXRT798S._edma_handle.tcdBase (*C var*), 316
drivers_RT_RT700_MIMXRT798S._edma_handle.tcdPool (*C var*), 316
drivers_RT_RT700_MIMXRT798S._edma_handle.tcdSize (*C var*), 316
drivers_RT_RT700_MIMXRT798S._edma_handle.tcdUsed (*C var*), 316
drivers_RT_RT700_MIMXRT798S._edma_handle.userData (*C var*), 315
drivers_RT_RT700_MIMXRT798S._edma_minor_offset_config (*C struct*), 312
drivers_RT_RT700_MIMXRT798S._edma_minor_offset_config.enableDestMinorOffset (*C var*), 312
drivers_RT_RT700_MIMXRT798S._edma_minor_offset_config.enableSrcMinorOffset (*C var*), 312
drivers_RT_RT700_MIMXRT798S._edma_minor_offset_config.minorOffset (*C var*), 312
drivers_RT_RT700_MIMXRT798S._edma_modulo (*C enum*), 302
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo1Gbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo1Kbytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo1Mbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo2bytes (*C enumerator*), 302
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo2Gbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo2Kbytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo2Mbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo4bytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo4Kbytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo4Mbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo8bytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo8Kbytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo8Mbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo16bytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo16Kbytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo16Mbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo32bytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo32Kbytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo32Mbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo64bytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo64Kbytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo64Mbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo128bytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo128Kbytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo128Mbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo256bytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo256Kbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo256Mbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo512bytes (*C enumerator*), 303
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo512Kbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_Modulo512Mbytes (*C enumerator*), 304
drivers_RT_RT700_MIMXRT798S._edma_modulo.kEDMA_ModuloDisable (*C enumerator*), 302
drivers_RT_RT700_MIMXRT798S._edma_tcd_type (*C enum*), 316
drivers_RT_RT700_MIMXRT798S._edma_tcd_type.kEDMA_EDMA4Flag (*C enumerator*), 316
drivers_RT_RT700_MIMXRT798S._edma_tcd_type.kEDMA_EDMA5Flag (*C enumerator*), 316
drivers_RT_RT700_MIMXRT798S._edma_transfer_config (*C struct*), 313
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.destAddr (*C var*), 313
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.destOffset (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.destTransferSize (*C var*), 313
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.dstAddrModulo (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.dstMajorLoopOffset (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.enableChannelMajorLoopLink (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.enableChannelMinorLoopLink (*C var*), 315
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.enabledInterruptMask (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.enableDstMinorLoopOffset (*C var*), 314

drivers_RT_RT700_MIMXRT798S._edma_transfer_config.enableSrcMinorLoopOffset (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.linkTCD (*C var*), 315
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.majorLoopCounts (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.majorLoopLinkChannel (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.minorLoopBytes (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.minorLoopLinkChannel (*C var*), 315
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.minorLoopOffset (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.srcAddr (*C var*), 313
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.srcAddrModulo (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.srcMajorLoopOffset (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.srcOffset (*C var*), 314
drivers_RT_RT700_MIMXRT798S._edma_transfer_config.srcTransferSize (*C var*), 313
drivers_RT_RT700_MIMXRT798S._edma_transfer_size (*C enum*), 302
drivers_RT_RT700_MIMXRT798S._edma_transfer_size.kEDMA_TransferSize1Bytes (*C enumerator*), 302
drivers_RT_RT700_MIMXRT798S._edma_transfer_size.kEDMA_TransferSize2Bytes (*C enumerator*), 302
drivers_RT_RT700_MIMXRT798S._edma_transfer_size.kEDMA_TransferSize4Bytes (*C enumerator*), 302
drivers_RT_RT700_MIMXRT798S._edma_transfer_size.kEDMA_TransferSize8Bytes (*C enumerator*), 302
drivers_RT_RT700_MIMXRT798S._edma_transfer_size.kEDMA_TransferSize16Bytes (*C enumerator*), 302
drivers_RT_RT700_MIMXRT798S._edma_transfer_size.kEDMA_TransferSize32Bytes (*C enumerator*), 302
drivers_RT_RT700_MIMXRT798S._edma_transfer_size.kEDMA_TransferSize64Bytes (*C enumerator*), 302
drivers_RT_RT700_MIMXRT798S._edma_transfer_size.kEDMA_TransferSize128Bytes (*C enumerator*), 302
drivers_RT_RT700_MIMXRT798S._edma_transfer_type (*C enum*), 306
drivers_RT_RT700_MIMXRT798S._edma_transfer_type.kEDMA_MemoryToMemory (*C enumerator*), 306
drivers_RT_RT700_MIMXRT798S._edma_transfer_type.kEDMA_MemoryToPeripheral (*C enumerator*), 306
drivers_RT_RT700_MIMXRT798S._edma_transfer_type.kEDMA_PeripheralToMemory (*C enumerator*), 306
drivers_RT_RT700_MIMXRT798S._edma_transfer_type.kEDMA_PeripheralToPeripheral (*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan (*C enum*), 326
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan0 (*C enumerator*), 326
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan1 (*C enumerator*), 326
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan2 (*C enumerator*), 326
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan3 (*C enumerator*), 326
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan4 (*C enumerator*), 326
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan5 (*C enumerator*), 327
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan6 (*C enumerator*), 327
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan7 (*C enumerator*), 327
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan8 (*C enumerator*), 327
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan9 (*C enumera-*

tor), 327
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan10 (*C enumerator*), 327
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan11 (*C enumerator*), 327
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan12 (*C enumerator*), 327
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan13 (*C enumerator*), 327
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan14 (*C enumerator*), 327
drivers_RT_RT700_MIMXRT798S._ezhv2arm_int_chan.kEZHV_EzhvToArmIntChan15 (*C enumerator*), 327
drivers_RT_RT700_MIMXRT798S._ezhv_copy_image (*C struct*), 330
drivers_RT_RT700_MIMXRT798S._ezhv_param (*C struct*), 330
drivers_RT_RT700_MIMXRT798S._FLASH_run_context_t (*C union*), 840
drivers_RT_RT700_MIMXRT798S._FLASH_run_context_t.B (*C struct*), 841
drivers_RT_RT700_MIMXRT798S._FLASH_run_context_t.B.current_mode (*C var*), 841
drivers_RT_RT700_MIMXRT798S._FLASH_run_context_t.B.reserved1 (*C var*), 841
drivers_RT_RT700_MIMXRT798S._FLASH_run_context_t.B.reseverd0 (*C var*), 841
drivers_RT_RT700_MIMXRT798S._FLASH_run_context_t.B.restore_sequence (*C var*), 841
drivers_RT_RT700_MIMXRT798S._flexio_camera_config (*C struct*), 333
drivers_RT_RT700_MIMXRT798S._flexio_camera_config.enablecamera (*C var*), 333
drivers_RT_RT700_MIMXRT798S._flexio_camera_config.enableFastAccess (*C var*), 333
drivers_RT_RT700_MIMXRT798S._flexio_camera_config.enableInDebug (*C var*), 333
drivers_RT_RT700_MIMXRT798S._flexio_camera_config.enableInDoze (*C var*), 333
drivers_RT_RT700_MIMXRT798S._flexio_camera_status_flags (*C enum*), 332
drivers_RT_RT700_MIMXRT798S._flexio_camera_status_flags.kFLEXIO_CAMERA_RxDataRegFullFlag (*C enumerator*), 332
drivers_RT_RT700_MIMXRT798S._flexio_camera_status_flags.kFLEXIO_CAMERA_RxErrorFlag (*C enumerator*), 332
drivers_RT_RT700_MIMXRT798S._flexio_camera_transfer (*C struct*), 334
drivers_RT_RT700_MIMXRT798S._flexio_camera_transfer.dataAddress (*C var*), 334
drivers_RT_RT700_MIMXRT798S._flexio_camera_transfer.dataNum (*C var*), 334
drivers_RT_RT700_MIMXRT798S._flexio_camera_type (*C struct*), 333
drivers_RT_RT700_MIMXRT798S._flexio_camera_type.datPinStartIdx (*C var*), 333
drivers_RT_RT700_MIMXRT798S._flexio_camera_type.flexioBase (*C var*), 333
drivers_RT_RT700_MIMXRT798S._flexio_camera_type.hrefPinIdx (*C var*), 333
drivers_RT_RT700_MIMXRT798S._flexio_camera_type.pclkPinIdx (*C var*), 333
drivers_RT_RT700_MIMXRT798S._flexio_camera_type.shifterCount (*C var*), 333
drivers_RT_RT700_MIMXRT798S._flexio_camera_type.shifterStartIdx (*C var*), 333
drivers_RT_RT700_MIMXRT798S._flexio_camera_type.timerIdx (*C var*), 333
drivers_RT_RT700_MIMXRT798S._flexio_config_ (*C struct*), 351
drivers_RT_RT700_MIMXRT798S._flexio_config_.enableFastAccess (*C var*), 351
drivers_RT_RT700_MIMXRT798S._flexio_config_.enableFlexio (*C var*), 351
drivers_RT_RT700_MIMXRT798S._flexio_config_.enableInDebug (*C var*), 351
drivers_RT_RT700_MIMXRT798S._flexio_config_.enableInDoze (*C var*), 351
drivers_RT_RT700_MIMXRT798S._flexio_gpio_config (*C struct*), 353
drivers_RT_RT700_MIMXRT798S._flexio_gpio_config.inputConfig (*C var*), 354
drivers_RT_RT700_MIMXRT798S._flexio_gpio_config.outputLogic (*C var*), 354
drivers_RT_RT700_MIMXRT798S._flexio_gpio_config.pinDirection (*C var*), 354
drivers_RT_RT700_MIMXRT798S._flexio_gpio_direction (*C enum*), 348
drivers_RT_RT700_MIMXRT798S._flexio_gpio_direction.kFLEXIO_DigitalInput (*C enumerator*), 348
drivers_RT_RT700_MIMXRT798S._flexio_gpio_direction.kFLEXIO_DigitalOutput (*C enumerator*), 348
drivers_RT_RT700_MIMXRT798S._flexio_i2c_direction (*C enum*), 374
drivers_RT_RT700_MIMXRT798S._flexio_i2c_direction.kFLEXIO_I2C_Read (*C enumerator*), 374

drivers_RT_RT700_MIMXRT798S._flexio_i2c_direction.kFLEXIO_I2C_Write (*C enumerator*), 374
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_config (*C struct*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_config.baudRate_Bps (*C var*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_config.enableFastAccess (*C var*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_config.enableInDebug (*C var*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_config.enableInDoze (*C var*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_config.enableMaster (*C var*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_handle (*C struct*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_handle.completionCallback (*C var*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_handle.needRestart (*C var*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_handle.state (*C var*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_handle.transfer (*C var*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_handle.transferSize (*C var*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_handle.userData (*C var*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_interrupt (*C enum*), 373
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_interrupt.kFLEXIO_I2C_RxFullInterruptEnable (*C enumerator*), 373
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_interrupt.kFLEXIO_I2C_TxEmptyInterruptEnable (*C enumerator*), 373
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_status_flags (*C enum*), 373
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_status_flags.kFLEXIO_I2C_ReceiveNakFlag (*C enumerator*), 374
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_status_flags.kFLEXIO_I2C_RxFullFlag (*C enumerator*), 374
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_status_flags.kFLEXIO_I2C_TxEmptyFlag (*C enumerator*), 374
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_transfer (*C struct*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_transfer.data (*C var*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_transfer.dataSize (*C var*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_transfer.direction (*C var*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_transfer.flags (*C var*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_transfer.slaveAddress (*C var*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_transfer.subaddress (*C var*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_master_transfer.subaddressSize (*C var*), 376
drivers_RT_RT700_MIMXRT798S._flexio_i2c_type (*C struct*), 374
drivers_RT_RT700_MIMXRT798S._flexio_i2c_type.baudrate (*C var*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_type.flexioBase (*C var*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_type.SCLPinIndex (*C var*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_type.SDAPinIndex (*C var*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_type.shifterIndex (*C var*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2c_type.timerIndex (*C var*), 375
drivers_RT_RT700_MIMXRT798S._flexio_i2s_config (*C struct*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_config.bclkPinPolarity (*C var*), 387
drivers_RT_RT700_MIMXRT798S._flexio_i2s_config.enableI2S (*C var*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_config.fsPinPolarity (*C var*), 387
drivers_RT_RT700_MIMXRT798S._flexio_i2s_config.masterSlave (*C var*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_config.rxPinPolarity (*C var*), 387
drivers_RT_RT700_MIMXRT798S._flexio_i2s_config.rxTimerPolarity (*C var*), 387
drivers_RT_RT700_MIMXRT798S._flexio_i2s_config.txPinPolarity (*C var*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_config.txTimerPolarity (*C var*), 387
drivers_RT_RT700_MIMXRT798S._flexio_i2s_edma_handle (*C struct*), 357
drivers_RT_RT700_MIMXRT798S._flexio_i2s_edma_handle.bytesPerFrame (*C var*), 357
drivers_RT_RT700_MIMXRT798S._flexio_i2s_edma_handle.callback (*C var*), 357
drivers_RT_RT700_MIMXRT798S._flexio_i2s_edma_handle.dmaHandle (*C var*), 357
drivers_RT_RT700_MIMXRT798S._flexio_i2s_edma_handle.nbytes (*C var*), 357
drivers_RT_RT700_MIMXRT798S._flexio_i2s_edma_handle.queue (*C var*), 357
drivers_RT_RT700_MIMXRT798S._flexio_i2s_edma_handle.queueDriver (*C var*), 357
drivers_RT_RT700_MIMXRT798S._flexio_i2s_edma_handle.queueUser (*C var*), 357

drivers_RT_RT700_MIMXRT798S._flexio_i2s_edma_handle.state (*C var*), 357
drivers_RT_RT700_MIMXRT798S._flexio_i2s_edma_handle.tcd (*C var*), 357
drivers_RT_RT700_MIMXRT798S._flexio_i2s_edma_handle.transferSize (*C var*), 357
drivers_RT_RT700_MIMXRT798S._flexio_i2s_edma_handle.userData (*C var*), 357
drivers_RT_RT700_MIMXRT798S._flexio_i2s_format (*C struct*), 387
drivers_RT_RT700_MIMXRT798S._flexio_i2s_format.bitWidth (*C var*), 387
drivers_RT_RT700_MIMXRT798S._flexio_i2s_format.sampleRate_Hz (*C var*), 387
drivers_RT_RT700_MIMXRT798S._flexio_i2s_handle (*C struct*), 387
drivers_RT_RT700_MIMXRT798S._flexio_i2s_handle.bitWidth (*C var*), 388
drivers_RT_RT700_MIMXRT798S._flexio_i2s_handle.callback (*C var*), 388
drivers_RT_RT700_MIMXRT798S._flexio_i2s_handle.queue (*C var*), 388
drivers_RT_RT700_MIMXRT798S._flexio_i2s_handle.queueDriver (*C var*), 388
drivers_RT_RT700_MIMXRT798S._flexio_i2s_handle.queueUser (*C var*), 388
drivers_RT_RT700_MIMXRT798S._flexio_i2s_handle.state (*C var*), 388
drivers_RT_RT700_MIMXRT798S._flexio_i2s_handle.transferSize (*C var*), 388
drivers_RT_RT700_MIMXRT798S._flexio_i2s_handle.userData (*C var*), 388
drivers_RT_RT700_MIMXRT798S._flexio_i2s_master_slave (*C enum*), 383
drivers_RT_RT700_MIMXRT798S._flexio_i2s_master_slave.kFLEXIO_I2S_Master (*C enumerator*), 383
drivers_RT_RT700_MIMXRT798S._flexio_i2s_master_slave.kFLEXIO_I2S_Slave (*C enumerator*), 383
drivers_RT_RT700_MIMXRT798S._flexio_i2s_sample_rate (*C enum*), 384
drivers_RT_RT700_MIMXRT798S._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate8KHz (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate12KHz (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate16KHz (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate24KHz (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate32KHz (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate48KHz (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate96KHz (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate11025Hz (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate22050Hz (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S._flexio_i2s_sample_rate.kFLEXIO_I2S_SampleRate44100Hz (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S._flexio_i2s_transfer (*C struct*), 387
drivers_RT_RT700_MIMXRT798S._flexio_i2s_transfer.data (*C var*), 387
drivers_RT_RT700_MIMXRT798S._flexio_i2s_transfer.dataSize (*C var*), 387
drivers_RT_RT700_MIMXRT798S._flexio_i2s_type (*C struct*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_type.bclkPinIndex (*C var*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_type.bclkTimerIndex (*C var*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_type.flexioBase (*C var*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_type.fsPinIndex (*C var*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_type.fsTimerIndex (*C var*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_type.rxPinIndex (*C var*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_type.rxShifterIndex (*C var*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_type.txPinIndex (*C var*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_type.txShifterIndex (*C var*), 386
drivers_RT_RT700_MIMXRT798S._flexio_i2s_word_width (*C enum*), 385
drivers_RT_RT700_MIMXRT798S._flexio_i2s_word_width.kFLEXIO_I2S_WordWidth8bits (*C enumerator*), 385

drivers_RT_RT700_MIMXRT798S._flexio_i2s_word_width.kFLEXIO_I2S_WordWidth16bits (*C enumerator*), 385
drivers_RT_RT700_MIMXRT798S._flexio_i2s_word_width.kFLEXIO_I2S_WordWidth24bits (*C enumerator*), 385
drivers_RT_RT700_MIMXRT798S._flexio_i2s_word_width.kFLEXIO_I2S_WordWidth32bits (*C enumerator*), 385
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_bus (*C enum*), 396
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_bus.kFLEXIO_MCULCD_6800 (*C enumerator*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_bus.kFLEXIO_MCULCD_8080 (*C enumerator*), 396
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_config (*C struct*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_config.baudRate_Bps (*C var*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_config.enable (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_config.enableFastAccess (*C var*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_config.enableInDebug (*C var*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_config.enableInDoze (*C var*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_dma_enable (*C enum*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_dma_enable.kFLEXIO_MCULCD_RxDmaEnable (*C enumerator*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_dma_enable.kFLEXIO_MCULCD_TxDmaEnable (*C enumerator*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle (*C struct*), 359
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.base (*C var*), 359
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.completionCallback (*C var*), 360
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.dataAddrOrSameValue (*C var*), 360
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.dataCount (*C var*), 360
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.minorLoopBytes (*C var*), 359
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.remainingCount (*C var*), 360
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.rxDmaHandle (*C var*), 360
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.rxEdmaModulo (*C var*), 360
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.rxShifterNum (*C var*), 359
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.state (*C var*), 360
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.txDmaHandle (*C var*), 360
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.txEdmaModulo (*C var*), 360
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.txShifterNum (*C var*), 359
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_edma_handle.userData (*C var*), 360
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_handle (*C struct*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_handle.completionCallback (*C var*), 401
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_handle.dataAddrOrSameValue (*C var*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_handle.dataCount (*C var*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_handle.remainingCount (*C var*), 401
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_handle.state (*C var*), 401
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_handle.userData (*C var*), 401
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_interrupt_enable (*C enum*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_interrupt_enable.kFLEXIO_MCULCD_RxFullInterruptEnable (*C enumerator*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_interrupt_enable.kFLEXIO_MCULCD_TxEmptyInterruptEnabl (*C enumerator*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_pixel_format (*C enum*), 396
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_pixel_format.kFLEXIO_MCULCD_BGR565 (*C enumerator*), 396
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_pixel_format.kFLEXIO_MCULCD_BGR888 (*C enumerator*), 396
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_pixel_format.kFLEXIO_MCULCD_RGB565 (*C enumerator*), 396

drivers_RT_RT700_MIMXRT798S._flexio_mculcd_pixel_format.kFLEXIO_MCULCD_RGB888 (*C enumerator*), 396
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_status_flags (*C enum*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_status_flags.kFLEXIO_MCULCD_RxFullFlag (*C enumerator*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_status_flags.kFLEXIO_MCULCD_TxEmptyFlag (*C enumerator*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_transfer (*C struct*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_transfer_mode (*C enum*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_transfer_mode.kFLEXIO_MCULCD_ReadArray (*C enumerator*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_transfer_mode.kFLEXIO_MCULCD_WriteArray (*C enumerator*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_transfer_mode.kFLEXIO_MCULCD_WriteSameValue (*C enumerator*), 397
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_transfer.command (*C var*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_transfer.dataAddrOrSameValue (*C var*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_transfer.dataOnly (*C var*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_transfer.dataSize (*C var*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_transfer.mode (*C var*), 400
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type (*C struct*), 398
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.busType (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.dataPinStartIndex (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.ENWRPinIndex (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.flexioBase (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.RDPinIndex (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.rxShifterEndIndex (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.rxShifterStartIndex (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.setCSPin (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.setRDWRPin (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.setRSPin (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.timerIndex (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.txShifterEndIndex (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_mculcd_type.txShifterStartIndex (*C var*), 399
drivers_RT_RT700_MIMXRT798S._flexio_pin_config (*C enum*), 342
drivers_RT_RT700_MIMXRT798S._flexio_pin_config.kFLEXIO_PinConfigBidirectionOutputData (*C enumerator*), 342
drivers_RT_RT700_MIMXRT798S._flexio_pin_config.kFLEXIO_PinConfigOpenDrainOrBidirection (*C enumerator*), 342
drivers_RT_RT700_MIMXRT798S._flexio_pin_config.kFLEXIO_PinConfigOutput (*C enumerator*), 343
drivers_RT_RT700_MIMXRT798S._flexio_pin_config.kFLEXIO_PinConfigOutputDisabled (*C enumerator*), 342
drivers_RT_RT700_MIMXRT798S._flexio_pin_input_config (*C enum*), 348
drivers_RT_RT700_MIMXRT798S._flexio_pin_input_config.kFLEXIO_FlagFallingEdgeEnable (*C enumerator*), 349
drivers_RT_RT700_MIMXRT798S._flexio_pin_input_config.kFLEXIO_FlagRisingEdgeEnable (*C enumerator*), 349
drivers_RT_RT700_MIMXRT798S._flexio_pin_input_config.kFLEXIO_InputInterruptDisabled (*C enumerator*), 348
drivers_RT_RT700_MIMXRT798S._flexio_pin_input_config.kFLEXIO_InputInterruptEnable (*C enumerator*), 349
drivers_RT_RT700_MIMXRT798S._flexio_pin_polarity (*C enum*), 343
drivers_RT_RT700_MIMXRT798S._flexio_pin_polarity.kFLEXIO_PinActiveHigh (*C enumerator*), 343
drivers_RT_RT700_MIMXRT798S._flexio_pin_polarity.kFLEXIO_PinActiveLow (*C enumerator*), 343
drivers_RT_RT700_MIMXRT798S._flexio_shifter_buffer_type (*C enum*), 348

drivers_RT_RT700_MIMXRT798S._flexio_shifter_buffer_type.kFLEXIO_ShifterBuffer (*C enumerator*), 348
drivers_RT_RT700_MIMXRT798S._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferBitByteSwapped (*C enumerator*), 348
drivers_RT_RT700_MIMXRT798S._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferBitSwapped (*C enumerator*), 348
drivers_RT_RT700_MIMXRT798S._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferByteSwapped (*C enumerator*), 348
drivers_RT_RT700_MIMXRT798S._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferHalfWordSwapped (*C enumerator*), 348
drivers_RT_RT700_MIMXRT798S._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferNibbleByteSwapped (*C enumerator*), 348
drivers_RT_RT700_MIMXRT798S._flexio_shifter_buffer_type.kFLEXIO_ShifterBufferNibbleSwapped (*C enumerator*), 348
drivers_RT_RT700_MIMXRT798S._flexio_shifter_config (*C struct*), 353
drivers_RT_RT700_MIMXRT798S._flexio_shifter_config.inputSource (*C var*), 353
drivers_RT_RT700_MIMXRT798S._flexio_shifter_config.parallelWidth (*C var*), 353
drivers_RT_RT700_MIMXRT798S._flexio_shifter_config.pinConfig (*C var*), 353
drivers_RT_RT700_MIMXRT798S._flexio_shifter_config.pinPolarity (*C var*), 353
drivers_RT_RT700_MIMXRT798S._flexio_shifter_config.pinSelect (*C var*), 353
drivers_RT_RT700_MIMXRT798S._flexio_shifter_config.shifterMode (*C var*), 353
drivers_RT_RT700_MIMXRT798S._flexio_shifter_config.shifterStart (*C var*), 353
drivers_RT_RT700_MIMXRT798S._flexio_shifter_config.shifterStop (*C var*), 353
drivers_RT_RT700_MIMXRT798S._flexio_shifter_config.timerPolarity (*C var*), 353
drivers_RT_RT700_MIMXRT798S._flexio_shifter_config.timerSelect (*C var*), 353
drivers_RT_RT700_MIMXRT798S._flexio_shifter_input_source (*C enum*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_input_source.kFLEXIO_ShifterInputFromNextShifterOutput (*C enumerator*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_input_source.kFLEXIO_ShifterInputFromPin (*C enumerator*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_mode (*C enum*), 346
drivers_RT_RT700_MIMXRT798S._flexio_shifter_mode.kFLEXIO_ShifterDisabled (*C enumerator*), 346
drivers_RT_RT700_MIMXRT798S._flexio_shifter_mode.kFLEXIO_ShifterModeLogic (*C enumerator*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_mode.kFLEXIO_ShifterModeMatchContinuous (*C enumerator*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_mode.kFLEXIO_ShifterModeMatchStore (*C enumerator*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_mode.kFLEXIO_ShifterModeReceive (*C enumerator*), 346
drivers_RT_RT700_MIMXRT798S._flexio_shifter_mode.kFLEXIO_ShifterModeState (*C enumerator*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_mode.kFLEXIO_ShifterModeTransmit (*C enumerator*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_start_bit (*C enum*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_start_bit.kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable (*C enumerator*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_start_bit.kFLEXIO_ShifterStartBitDisabledLoadDataOnShift (*C enumerator*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_start_bit.kFLEXIO_ShifterStartBitHigh (*C enumerator*), 348
drivers_RT_RT700_MIMXRT798S._flexio_shifter_start_bit.kFLEXIO_ShifterStartBitLow (*C enumerator*), 348
drivers_RT_RT700_MIMXRT798S._flexio_shifter_stop_bit (*C enum*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_stop_bit.kFLEXIO_ShifterStopBitDisable (*C enumerator*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_stop_bit.kFLEXIO_ShifterStopBitHigh (*C enu-*

merator), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_stop_bit.kFLEXIO_ShifterStopBitLow (*C enumerator*), 347
drivers_RT_RT700_MIMXRT798S._flexio_shifter_timer_polarity (*C enum*), 346
drivers_RT_RT700_MIMXRT798S._flexio_shifter_timer_polarity.kFLEXIO_ShifterTimerPolarityOnNegitive (*C enumerator*), 346
drivers_RT_RT700_MIMXRT798S._flexio_shifter_timer_polarity.kFLEXIO_ShifterTimerPolarityOnPositive (*C enumerator*), 346
drivers_RT_RT700_MIMXRT798S._flexio_spi_clock_phase (*C enum*), 409
drivers_RT_RT700_MIMXRT798S._flexio_spi_clock_phase.kFLEXIO_SPI_ClockPhaseFirstEdge (*C enumerator*), 409
drivers_RT_RT700_MIMXRT798S._flexio_spi_clock_phase.kFLEXIO_SPI_ClockPhaseSecondEdge (*C enumerator*), 409
drivers_RT_RT700_MIMXRT798S._flexio_spi_data_bitcount_mode (*C enum*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_data_bitcount_mode.kFLEXIO_SPI_8BitMode (*C enumerator*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_data_bitcount_mode.kFLEXIO_SPI_16BitMode (*C enumerator*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_data_bitcount_mode.kFLEXIO_SPI_32BitMode (*C enumerator*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_dma_enable (*C enum*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_dma_enable.kFLEXIO_SPI_DmaAllEnable (*C enumerator*), 411
drivers_RT_RT700_MIMXRT798S._flexio_spi_dma_enable.kFLEXIO_SPI_RxDmaEnable (*C enumerator*), 411
drivers_RT_RT700_MIMXRT798S._flexio_spi_dma_enable.kFLEXIO_SPI_TxDmaEnable (*C enumerator*), 411
drivers_RT_RT700_MIMXRT798S._flexio_spi_interrupt_enable (*C enum*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_interrupt_enable.kFLEXIO_SPI_RxFullInterruptEnable (*C enumerator*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_interrupt_enable.kFLEXIO_SPI_TxEmptyInterruptEnable (*C enumerator*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_config (*C struct*), 413
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_config.baudRate_Bps (*C var*), 413
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_config.dataMode (*C var*), 413
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_config.enableFastAccess (*C var*), 413
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_config.enableInDebug (*C var*), 413
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_config.enableInDoze (*C var*), 413
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_config.enableMaster (*C var*), 413
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_config.phase (*C var*), 413
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_edma_handle (*C struct*), 363
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_edma_handle.callback (*C var*), 364
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_edma_handle.nbytes (*C var*), 363
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_edma_handle.rxHandle (*C var*), 363
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_edma_handle.rxInProgress (*C var*), 363
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_edma_handle.transferSize (*C var*), 363
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_edma_handle.txHandle (*C var*), 363
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_edma_handle.txInProgress (*C var*), 363
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_edma_handle.userData (*C var*), 364
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle (*C struct*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle.bytePerFrame (*C var*), 415
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle.callback (*C var*), 415
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle.direction (*C var*), 415
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle.isCsContinuous (*C var*), 415
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle.rxData (*C var*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle.rxRemainingBytes (*C var*), 415
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle.state (*C var*), 415
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle.timer1Cfg (*C var*), 415

drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle.transferSize (*C var*), 415
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle.txData (*C var*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle.txRemainingBytes (*C var*), 415
drivers_RT_RT700_MIMXRT798S._flexio_spi_master_handle.userData (*C var*), 415
drivers_RT_RT700_MIMXRT798S._flexio_spi_shift_direction (*C enum*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_shift_direction.kFLEXIO_SPI_LsbFirst (*C enumerator*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_shift_direction.kFLEXIO_SPI_MsbFirst (*C enumerator*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_slave_config (*C struct*), 413
drivers_RT_RT700_MIMXRT798S._flexio_spi_slave_config.dataMode (*C var*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_slave_config.enableFastAccess (*C var*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_slave_config.enableInDebug (*C var*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_slave_config.enableInDoze (*C var*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_slave_config.enableSlave (*C var*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_slave_config.phase (*C var*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_status_flags (*C enum*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_status_flags.kFLEXIO_SPI_RxBufferFullFlag (*C enumerator*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_status_flags.kFLEXIO_SPI_TxBufferEmptyFlag (*C enumerator*), 410
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer (*C struct*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer_flags (*C enum*), 411
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer_flags.kFLEXIO_SPI_8bitLsb (*C enumerator*), 411
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer_flags.kFLEXIO_SPI_8bitMsb (*C enumerator*), 411
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer_flags.kFLEXIO_SPI_16bitLsb (*C enumerator*), 411
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer_flags.kFLEXIO_SPI_16bitMsb (*C enumerator*), 411
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer_flags.kFLEXIO_SPI_32bitLsb (*C enumerator*), 411
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer_flags.kFLEXIO_SPI_32bitMsb (*C enumerator*), 411
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer_flags.kFLEXIO_SPI_csContinuous (*C enumerator*), 411
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer.dataSize (*C var*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer.flags (*C var*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer.rxData (*C var*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_transfer.txData (*C var*), 414
drivers_RT_RT700_MIMXRT798S._flexio_spi_type (*C struct*), 412
drivers_RT_RT700_MIMXRT798S._flexio_spi_type.CSnPinIndex (*C var*), 413
drivers_RT_RT700_MIMXRT798S._flexio_spi_type.flexioBase (*C var*), 412
drivers_RT_RT700_MIMXRT798S._flexio_spi_type.SCKPinIndex (*C var*), 413
drivers_RT_RT700_MIMXRT798S._flexio_spi_type.SDIPinIndex (*C var*), 412
drivers_RT_RT700_MIMXRT798S._flexio_spi_type.SDOPinIndex (*C var*), 412
drivers_RT_RT700_MIMXRT798S._flexio_spi_type.shifterIndex (*C var*), 413
drivers_RT_RT700_MIMXRT798S._flexio_spi_type.timerIndex (*C var*), 413
drivers_RT_RT700_MIMXRT798S._flexio_timer_config (*C struct*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.pinConfig (*C var*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.pinPolarity (*C var*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.pinSelect (*C var*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.timerCompare (*C var*), 353
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.timerDecrement (*C var*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.timerDisable (*C var*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.timerEnable (*C var*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.timerMode (*C var*), 352

drivers_RT_RT700_MIMXRT798S._flexio_timer_config.timerOutput (*C var*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.timerReset (*C var*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.timerStart (*C var*), 353
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.timerStop (*C var*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.triggerPolarity (*C var*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.triggerSelect (*C var*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_config.triggerSource (*C var*), 352
drivers_RT_RT700_MIMXRT798S._flexio_timer_decrement_source (*C enum*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_decrement_source.kFLEXIO_TimerDecSrcOnFlexIOClockShiftTim (*C enumerator*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_decrement_source.kFLEXIO_TimerDecSrcOnPinInputShiftPinInp (*C enumerator*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_decrement_source.kFLEXIO_TimerDecSrcOnTriggerInputShiftTim (*C enumerator*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_decrement_source.kFLEXIO_TimerDecSrcOnTriggerInputShiftTrig (*C enumerator*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_disable_condition (*C enum*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_disable_condition.kFLEXIO_TimerDisableNever (*C enumerator*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnPinBothEdge (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnPinBothEdgeTrigger (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnPreTimerDisable (*C enumerator*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnTimerCompare (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnTimerCompareTrigg (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_disable_condition.kFLEXIO_TimerDisableOnTriggerFallingEdge (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_enable_condition (*C enum*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_enable_condition.kFLEXIO_TimerEnabledAlways (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnPinRisingEdge (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnPinRisingEdgeTrigger (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnPrevTimerEnable (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnTriggerBothEdge (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnTriggerHigh (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnTriggerHighPinHigh (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_enable_condition.kFLEXIO_TimerEnableOnTriggerRisingEdge (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_mode (*C enum*), 343
drivers_RT_RT700_MIMXRT798S._flexio_timer_mode.kFLEXIO_TimerModeDisabled (*C enumerator*), 343
drivers_RT_RT700_MIMXRT798S._flexio_timer_mode.kFLEXIO_TimerModeDual8BitBaudBit (*C enumerator*), 343
drivers_RT_RT700_MIMXRT798S._flexio_timer_mode.kFLEXIO_TimerModeDual8BitPWM (*C enumerator*), 343
drivers_RT_RT700_MIMXRT798S._flexio_timer_mode.kFLEXIO_TimerModeDual8BitPWMLow (*C enumerator*), 343
drivers_RT_RT700_MIMXRT798S._flexio_timer_mode.kFLEXIO_TimerModeSingle16Bit (*C enu-*

merator), 343
drivers_RT_RT700_MIMXRT798S._flexio_timer_output (*C enum*), 343
drivers_RT_RT700_MIMXRT798S._flexio_timer_output_state (*C enum*), 346
drivers_RT_RT700_MIMXRT798S._flexio_timer_output_state.kFLEXIO_PwmHigh (*C enumerator*), 346
drivers_RT_RT700_MIMXRT798S._flexio_timer_output_state.kFLEXIO_PwmLow (*C enumerator*), 346
drivers_RT_RT700_MIMXRT798S._flexio_timer_output.kFLEXIO_TimerOutputOneAffectedByReset (*C enumerator*), 343
drivers_RT_RT700_MIMXRT798S._flexio_timer_output.kFLEXIO_TimerOutputOneNotAffectedByReset (*C enumerator*), 343
drivers_RT_RT700_MIMXRT798S._flexio_timer_output.kFLEXIO_TimerOutputZeroAffectedByReset (*C enumerator*), 343
drivers_RT_RT700_MIMXRT798S._flexio_timer_output.kFLEXIO_TimerOutputZeroNotAffectedByReset (*C enumerator*), 343
drivers_RT_RT700_MIMXRT798S._flexio_timer_reset_condition (*C enum*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_reset_condition.kFLEXIO_TimerResetNever (*C enumerator*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerPinEqualToTimer (*C enumerator*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerPinRisingEdge (*C enumerator*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerTriggerBothEdge (*C enumerator*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerTriggerEqualToTi (*C enumerator*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_reset_condition.kFLEXIO_TimerResetOnTimerTriggerRisingEdge (*C enumerator*), 344
drivers_RT_RT700_MIMXRT798S._flexio_timer_start_bit_condition (*C enum*), 346
drivers_RT_RT700_MIMXRT798S._flexio_timer_start_bit_condition.kFLEXIO_TimerStartBitDisabled (*C enumerator*), 346
drivers_RT_RT700_MIMXRT798S._flexio_timer_start_bit_condition.kFLEXIO_TimerStartBitEnabled (*C enumerator*), 346
drivers_RT_RT700_MIMXRT798S._flexio_timer_stop_bit_condition (*C enum*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_stop_bit_condition.kFLEXIO_TimerStopBitDisabled (*C enumerator*), 345
drivers_RT_RT700_MIMXRT798S._flexio_timer_stop_bit_condition.kFLEXIO_TimerStopBitEnableOnTimerComp (*C enumerator*), 346
drivers_RT_RT700_MIMXRT798S._flexio_timer_stop_bit_condition.kFLEXIO_TimerStopBitEnableOnTimerComp (*C enumerator*), 346
drivers_RT_RT700_MIMXRT798S._flexio_timer_stop_bit_condition.kFLEXIO_TimerStopBitEnableOnTimerDisab (*C enumerator*), 346
drivers_RT_RT700_MIMXRT798S._flexio_timer_trigger_polarity (*C enum*), 342
drivers_RT_RT700_MIMXRT798S._flexio_timer_trigger_polarity.kFLEXIO_TimerTriggerPolarityActiveHigh (*C enumerator*), 342
drivers_RT_RT700_MIMXRT798S._flexio_timer_trigger_polarity.kFLEXIO_TimerTriggerPolarityActiveLow (*C enumerator*), 342
drivers_RT_RT700_MIMXRT798S._flexio_timer_trigger_source (*C enum*), 342
drivers_RT_RT700_MIMXRT798S._flexio_timer_trigger_source.kFLEXIO_TimerTriggerSourceExternal (*C enumerator*), 342
drivers_RT_RT700_MIMXRT798S._flexio_timer_trigger_source.kFLEXIO_TimerTriggerSourceInternal (*C enumerator*), 342
drivers_RT_RT700_MIMXRT798S._flexio_uart_bit_count_per_char (*C enum*), 423
drivers_RT_RT700_MIMXRT798S._flexio_uart_bit_count_per_char.kFLEXIO_UART_7BitsPerChar (*C enumerator*), 423
drivers_RT_RT700_MIMXRT798S._flexio_uart_bit_count_per_char.kFLEXIO_UART_8BitsPerChar (*C enumerator*), 423
drivers_RT_RT700_MIMXRT798S._flexio_uart_bit_count_per_char.kFLEXIO_UART_9BitsPerChar

(*C enumerator*), 423
drivers_RT_RT700_MIMXRT798S._flexio_uart_config (*C struct*), 425
drivers_RT_RT700_MIMXRT798S._flexio_uart_config.baudRate_Bps (*C var*), 425
drivers_RT_RT700_MIMXRT798S._flexio_uart_config.bitCountPerChar (*C var*), 425
drivers_RT_RT700_MIMXRT798S._flexio_uart_config.enableFastAccess (*C var*), 425
drivers_RT_RT700_MIMXRT798S._flexio_uart_config.enableInDebug (*C var*), 425
drivers_RT_RT700_MIMXRT798S._flexio_uart_config.enableInDoze (*C var*), 425
drivers_RT_RT700_MIMXRT798S._flexio_uart_config.enableUart (*C var*), 425
drivers_RT_RT700_MIMXRT798S._flexio_uart_edma_handle (*C struct*), 366
drivers_RT_RT700_MIMXRT798S._flexio_uart_edma_handle.callback (*C var*), 366
drivers_RT_RT700_MIMXRT798S._flexio_uart_edma_handle.nbytes (*C var*), 366
drivers_RT_RT700_MIMXRT798S._flexio_uart_edma_handle.rxDataSizeAll (*C var*), 366
drivers_RT_RT700_MIMXRT798S._flexio_uart_edma_handle.rxEdmaHandle (*C var*), 366
drivers_RT_RT700_MIMXRT798S._flexio_uart_edma_handle.rxState (*C var*), 367
drivers_RT_RT700_MIMXRT798S._flexio_uart_edma_handle.txDataSizeAll (*C var*), 366
drivers_RT_RT700_MIMXRT798S._flexio_uart_edma_handle.txEdmaHandle (*C var*), 366
drivers_RT_RT700_MIMXRT798S._flexio_uart_edma_handle.txState (*C var*), 367
drivers_RT_RT700_MIMXRT798S._flexio_uart_edma_handle.userData (*C var*), 366
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle (*C struct*), 425
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.callback (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.rxData (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.rxDataSize (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.rxDataSizeAll (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.rxRingBuffer (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.rxRingBufferHead (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.rxRingBufferSize (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.rxRingBufferTail (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.rxState (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.txData (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.txDataSize (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.txDataSizeAll (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.txState (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_handle.userData (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_interrupt_enable (*C enum*), 423
drivers_RT_RT700_MIMXRT798S._flexio_uart_interrupt_enable.kFLEXIO_UART_RxDataRegFullInterruptEnable
(*C enumerator*), 423
drivers_RT_RT700_MIMXRT798S._flexio_uart_interrupt_enable.kFLEXIO_UART_TxDataRegEmptyInterruptEna
(*C enumerator*), 423
drivers_RT_RT700_MIMXRT798S._flexio_uart_status_flags (*C enum*), 424
drivers_RT_RT700_MIMXRT798S._flexio_uart_status_flags.kFLEXIO_UART_RxDataRegFullFlag
(*C enumerator*), 424
drivers_RT_RT700_MIMXRT798S._flexio_uart_status_flags.kFLEXIO_UART_RxOverRunFlag (*C enumerator*), 424
drivers_RT_RT700_MIMXRT798S._flexio_uart_status_flags.kFLEXIO_UART_TxDataRegEmptyFlag
(*C enumerator*), 424
drivers_RT_RT700_MIMXRT798S._flexio_uart_transfer (*C struct*), 425
drivers_RT_RT700_MIMXRT798S._flexio_uart_transfer.___unnamed73___ (*C union*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_transfer.___unnamed73___.data (*C var*), 426
drivers_RT_RT700_MIMXRT798S._flexio_uart_transfer.___unnamed73___.rxData (*C var*), 427
drivers_RT_RT700_MIMXRT798S._flexio_uart_transfer.___unnamed73___.txData (*C var*), 427
drivers_RT_RT700_MIMXRT798S._flexio_uart_transfer.dataSize (*C var*), 425
drivers_RT_RT700_MIMXRT798S._flexio_uart_type (*C struct*), 424
drivers_RT_RT700_MIMXRT798S._flexio_uart_type.flexioBase (*C var*), 424
drivers_RT_RT700_MIMXRT798S._flexio_uart_type.RxPinIndex (*C var*), 424
drivers_RT_RT700_MIMXRT798S._flexio_uart_type.shifterIndex (*C var*), 425
drivers_RT_RT700_MIMXRT798S._flexio_uart_type.timerIndex (*C var*), 425
drivers_RT_RT700_MIMXRT798S._flexio_uart_type.TxPinIndex (*C var*), 424
drivers_RT_RT700_MIMXRT798S._freq_measure_config (*C struct*), 610

drivers_RT_RT700_MIMXRT798S._freq_measure_config.enableContinuousMode (*C var*), 610
drivers_RT_RT700_MIMXRT798S._freq_measure_config.operateMode (*C var*), 610
drivers_RT_RT700_MIMXRT798S._freq_measure_config.operateModeAttribute (*C var*), 610
drivers_RT_RT700_MIMXRT798S._freqme_interrupt_enable (*C enum*), 608
drivers_RT_RT700_MIMXRT798S._freqme_interrupt_enable.kFREQME_OverflowInterruptEnable (*C enumerator*), 608
drivers_RT_RT700_MIMXRT798S._freqme_interrupt_enable.kFREQME_ReadyInterruptEnable (*C enumerator*), 609
drivers_RT_RT700_MIMXRT798S._freqme_interrupt_enable.kFREQME_UnderflowInterruptEnable (*C enumerator*), 608
drivers_RT_RT700_MIMXRT798S._freqme_interrupt_status_flags (*C enum*), 608
drivers_RT_RT700_MIMXRT798S._freqme_interrupt_status_flags.kFREQME_AllInterruptStatusFlags (*C enumerator*), 608
drivers_RT_RT700_MIMXRT798S._freqme_interrupt_status_flags.kFREQME_OverflowInterruptStatusFlag (*C enumerator*), 608
drivers_RT_RT700_MIMXRT798S._freqme_interrupt_status_flags.kFREQME_ReadyInterruptStatusFlag (*C enumerator*), 608
drivers_RT_RT700_MIMXRT798S._freqme_interrupt_status_flags.kFREQME_UnderflowInterruptStatusFlag (*C enumerator*), 608
drivers_RT_RT700_MIMXRT798S._freqme_mode_attribute (*C union*), 609
drivers_RT_RT700_MIMXRT798S._freqme_mode_attribute.pulsePolarity (*C var*), 610
drivers_RT_RT700_MIMXRT798S._freqme_mode_attribute.refClkScaleFactor (*C var*), 610
drivers_RT_RT700_MIMXRT798S._freqme_operate_mode (*C enum*), 609
drivers_RT_RT700_MIMXRT798S._freqme_operate_mode.kFREOME_PulseWidthMeasurementMode (*C enumerator*), 609
drivers_RT_RT700_MIMXRT798S._freqme_operate_mode.kFREQME_FreqMeasurementMode (*C enumerator*), 609
drivers_RT_RT700_MIMXRT798S._freqme_pulse_polarity (*C enum*), 609
drivers_RT_RT700_MIMXRT798S._freqme_pulse_polarity.kFREQME_PulseHighPeriod (*C enumerator*), 609
drivers_RT_RT700_MIMXRT798S._freqme_pulse_polarity.kFREQME_PulseLowPeriod (*C enumerator*), 609
drivers_RT_RT700_MIMXRT798S._gpio_checker_attribute (*C enum*), 434
drivers_RT_RT700_MIMXRT798S._gpio_checker_attribute.kGPIO_IgnoreAttributeCheck (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR (*C enumerator*), 434
drivers_RT_RT700_MIMXRT798S._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW (*C enumerator*), 434
drivers_RT_RT700_MIMXRT798S._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW (*C enumerator*), 434
drivers_RT_RT700_MIMXRT798S._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureR (*C enumerator*), 434
drivers_RT_RT700_MIMXRT798S._gpio_checker_attribute.kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW (*C enumerator*), 434
drivers_RT_RT700_MIMXRT798S._gpio_checker_attribute.kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureR (*C enumerator*), 434
drivers_RT_RT700_MIMXRT798S._gpio_checker_attribute.kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecure (*C enumerator*), 434
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config (*C enum*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_ActiveHighTriggerOutputEnable (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_ActiveLowTriggerOutputEnable (*C enumerator*), 436
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_DMAEitherEdge (*C enumerator*), 435

drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_DMAFallingEdge (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_DMARisingEdge (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_FlagEitherEdge (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_FlagFallingEdge (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_FlagRisingEdge (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_InterruptEitherEdge (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_InterruptFallingEdge (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_InterruptLogicOne (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_InterruptLogicZero (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_InterruptRisingEdge (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_config.kGPIO_InterruptStatusFlagDisabled (*C enumerator*), 435
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_selection (*C enum*), 436
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_selection.kGPIO_InterruptOutput0 (*C enumerator*), 436
drivers_RT_RT700_MIMXRT798S._gpio_interrupt_selection.kGPIO_InterruptOutput1 (*C enumerator*), 436
drivers_RT_RT700_MIMXRT798S._gpio_pin_config (*C struct*), 437
drivers_RT_RT700_MIMXRT798S._gpio_pin_config.outputLogic (*C var*), 437
drivers_RT_RT700_MIMXRT798S._gpio_pin_config.pinDirection (*C var*), 437
drivers_RT_RT700_MIMXRT798S._gpio_pin_direction (*C enum*), 434
drivers_RT_RT700_MIMXRT798S._gpio_pin_direction.kGPIO_DigitalInput (*C enumerator*), 434
drivers_RT_RT700_MIMXRT798S._gpio_pin_direction.kGPIO_DigitalOutput (*C enumerator*), 434
drivers_RT_RT700_MIMXRT798S._gpio_version_info (*C struct*), 437
drivers_RT_RT700_MIMXRT798S._gpio_version_info.feature (*C var*), 437
drivers_RT_RT700_MIMXRT798S._gpio_version_info.major (*C var*), 437
drivers_RT_RT700_MIMXRT798S._gpio_version_info.minor (*C var*), 437
drivers_RT_RT700_MIMXRT798S._i3c_baudrate (*C struct*), 476
drivers_RT_RT700_MIMXRT798S._i3c_baudrate.i2cBaud (*C var*), 476
drivers_RT_RT700_MIMXRT798S._i3c_baudrate.i3cOpenDrainBaud (*C var*), 476
drivers_RT_RT700_MIMXRT798S._i3c_baudrate.i3cPushPullBaud (*C var*), 476
drivers_RT_RT700_MIMXRT798S._i3c_bus_request (*C enum*), 469
drivers_RT_RT700_MIMXRT798S._i3c_bus_request.kI3C_RequestAutoIbi (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_bus_request.kI3C_RequestEmitStartAddr (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_bus_request.kI3C_RequestEmitStop (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_bus_request.kI3C_RequestForceExit (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_bus_request.kI3C_RequestIbiAckNack (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_bus_request.kI3C_RequestNone (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_bus_request.kI3C_RequestProcessDAA (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_bus_type (*C enum*), 470
drivers_RT_RT700_MIMXRT798S._i3c_bus_type.kI3C_TypeI2C (*C enumerator*), 470
drivers_RT_RT700_MIMXRT798S._i3c_bus_type.kI3C_TypeI3CDdr (*C enumerator*), 470
drivers_RT_RT700_MIMXRT798S._i3c_bus_type.kI3C_TypeI3CSdr (*C enumerator*), 470
drivers_RT_RT700_MIMXRT798S._i3c_config (*C struct*), 447
drivers_RT_RT700_MIMXRT798S._i3c_config.baudRate_Hz (*C var*), 447
drivers_RT_RT700_MIMXRT798S._i3c_config.bcr (*C var*), 448

drivers_RT_RT700_MIMXRT798S._i3c_config.dcr (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.disableTimeout (*C var*), 447
drivers_RT_RT700_MIMXRT798S._i3c_config.enableMaster (*C var*), 447
drivers_RT_RT700_MIMXRT798S._i3c_config.enableOpenDrainHigh (*C var*), 447
drivers_RT_RT700_MIMXRT798S._i3c_config.enableOpenDrainStop (*C var*), 447
drivers_RT_RT700_MIMXRT798S._i3c_config.enableSlave (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.hdrMode (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.hKeep (*C var*), 447
drivers_RT_RT700_MIMXRT798S._i3c_config.ignoreS0S1Error (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.masterDynamicAddress (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.matchSlaveStartStop (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.maxReadLength (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.maxWriteLength (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.nakAllRequest (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.offline (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.partNumber (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.restartSclDelay (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.startSclDelay (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.staticAddr (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_config.vendorID (*C var*), 448
drivers_RT_RT700_MIMXRT798S._i3c_device_info (*C struct*), 445
drivers_RT_RT700_MIMXRT798S._i3c_device_info.bcr (*C var*), 445
drivers_RT_RT700_MIMXRT798S._i3c_device_info.dcr (*C var*), 445
drivers_RT_RT700_MIMXRT798S._i3c_device_info.dynamicAddr (*C var*), 445
drivers_RT_RT700_MIMXRT798S._i3c_device_info.hdrMode (*C var*), 446
drivers_RT_RT700_MIMXRT798S._i3c_device_info.maxReadLength (*C var*), 446
drivers_RT_RT700_MIMXRT798S._i3c_device_info.maxWriteLength (*C var*), 446
drivers_RT_RT700_MIMXRT798S._i3c_device_info.partNumber (*C var*), 446
drivers_RT_RT700_MIMXRT798S._i3c_device_info.staticAddr (*C var*), 445
drivers_RT_RT700_MIMXRT798S._i3c_device_info.vendorID (*C var*), 445
drivers_RT_RT700_MIMXRT798S._i3c_direction (*C enum*), 471
drivers_RT_RT700_MIMXRT798S._i3c_direction.kI3C_Read (*C enumerator*), 471
drivers_RT_RT700_MIMXRT798S._i3c_direction.kI3C_Write (*C enumerator*), 471
drivers_RT_RT700_MIMXRT798S._i3c_hdr_mode (*C enum*), 445
drivers_RT_RT700_MIMXRT798S._i3c_hdr_mode.kI3C_HDRModeDDR (*C enumerator*), 445
drivers_RT_RT700_MIMXRT798S._i3c_hdr_mode.kI3C_HDRModeNone (*C enumerator*), 445
drivers_RT_RT700_MIMXRT798S._i3c_hdr_mode.kI3C_HDRModeTSL (*C enumerator*), 445
drivers_RT_RT700_MIMXRT798S._i3c_hdr_mode.kI3C_HDRModeTSP (*C enumerator*), 445
drivers_RT_RT700_MIMXRT798S._i3c_ibi_response (*C enum*), 470
drivers_RT_RT700_MIMXRT798S._i3c_ibi_response.kI3C_IbiRespAck (*C enumerator*), 470
drivers_RT_RT700_MIMXRT798S._i3c_ibi_response.kI3C_IbiRespAckMandatory (*C enumerator*), 470
drivers_RT_RT700_MIMXRT798S._i3c_ibi_response.kI3C_IbiRespManual (*C enumerator*), 470
drivers_RT_RT700_MIMXRT798S._i3c_ibi_response.kI3C_IbiRespNack (*C enumerator*), 470
drivers_RT_RT700_MIMXRT798S._i3c_ibi_state (*C enum*), 470
drivers_RT_RT700_MIMXRT798S._i3c_ibi_state.kI3C_IbiAckNackPending (*C enumerator*), 471
drivers_RT_RT700_MIMXRT798S._i3c_ibi_state.kI3C_IbiDataBuffNeed (*C enumerator*), 471
drivers_RT_RT700_MIMXRT798S._i3c_ibi_state.kI3C_IbiReady (*C enumerator*), 470
drivers_RT_RT700_MIMXRT798S._i3c_ibi_type (*C enum*), 470
drivers_RT_RT700_MIMXRT798S._i3c_ibi_type.kI3C_IbiHotJoin (*C enumerator*), 470
drivers_RT_RT700_MIMXRT798S._i3c_ibi_type.kI3C_IbiMasterRequest (*C enumerator*), 470
drivers_RT_RT700_MIMXRT798S._i3c_ibi_type.kI3C_IbiNormal (*C enumerator*), 470
drivers_RT_RT700_MIMXRT798S._i3c_legacy_i2c_speed_mode (*C enum*), 472
drivers_RT_RT700_MIMXRT798S._i3c_legacy_i2c_speed_mode.kI3C_LegacyI2CFastMode (*C enumerator*), 473
drivers_RT_RT700_MIMXRT798S._i3c_legacy_i2c_speed_mode.kI3C_LegacyI2CFastModePlus (*C enumerator*), 473

drivers_RT_RT700_MIMXRT798S._i3c_legacy_i2c_speed_mode.kI3C_LegacyI2CStandardMode (*C enumerator*), 472
drivers_RT_RT700_MIMXRT798S._i3c_master_baudrate_config (*C struct*), 476
drivers_RT_RT700_MIMXRT798S._i3c_master_baudrate_config.busMode (*C var*), 476
drivers_RT_RT700_MIMXRT798S._i3c_master_baudrate_config.enableODHPP (*C var*), 477
drivers_RT_RT700_MIMXRT798S._i3c_master_baudrate_config.i2cBaud (*C var*), 476
drivers_RT_RT700_MIMXRT798S._i3c_master_baudrate_config.i3cOpenDrainBaud (*C var*), 477
drivers_RT_RT700_MIMXRT798S._i3c_master_baudrate_config.i3cPushPullBaud (*C var*), 476
drivers_RT_RT700_MIMXRT798S._i3c_master_bus_mode (*C enum*), 472
drivers_RT_RT700_MIMXRT798S._i3c_master_bus_mode.kI3C_LimitedBus (*C enumerator*), 472
drivers_RT_RT700_MIMXRT798S._i3c_master_bus_mode.kI3C_MixedFastBus (*C enumerator*), 472
drivers_RT_RT700_MIMXRT798S._i3c_master_bus_mode.kI3C_PureBus (*C enumerator*), 472
drivers_RT_RT700_MIMXRT798S._i3c_master_config (*C struct*), 477
drivers_RT_RT700_MIMXRT798S._i3c_master_config.baudRate_Hz (*C var*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_config.disableTimeout (*C var*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_config.enableMaster (*C var*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_config.enableOpenDrainHigh (*C var*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_config.enableOpenDrainStop (*C var*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_config.hKeep (*C var*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_config.restartSclDelay (*C var*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_config.startSclDelay (*C var*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_daa_baudrate (*C struct*), 476
drivers_RT_RT700_MIMXRT798S._i3c_master_daa_baudrate.i3cOpenDrainBaud (*C var*), 476
drivers_RT_RT700_MIMXRT798S._i3c_master_daa_baudrate.i3cPushPullBaud (*C var*), 476
drivers_RT_RT700_MIMXRT798S._i3c_master_daa_baudrate.sourceClock_Hz (*C var*), 476
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_callback (*C struct*), 482
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_callback.ibiCallback (*C var*), 482
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_callback.slave2Master (*C var*), 482
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_callback.transferComplete (*C var*), 482
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle (*C struct*), 482
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.base (*C var*), 482
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.callback (*C var*), 483
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.ibiAddress (*C var*), 483
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.ibiBuff (*C var*), 483
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.ibiFlag (*C var*), 483
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.ibiPayloadSize (*C var*), 483
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.ibiType (*C var*), 483
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.result (*C var*), 483
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.rxDmaHandle (*C var*), 483
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.state (*C var*), 482
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.subaddressBuffer (*C var*), 482
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.subaddressCount (*C var*), 482
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.transfer (*C var*), 482
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.transferCount (*C var*), 482
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.txDmaHandle (*C var*), 483
drivers_RT_RT700_MIMXRT798S._i3c_master_edma_handle.userData (*C var*), 483
drivers_RT_RT700_MIMXRT798S._i3c_master_enable (*C enum*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_enable.kI3C_MasterCapable (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_master_enable.kI3C_MasterOff (*C enumerator*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_enable.kI3C_MasterOn (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_master_error_flags (*C enum*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_error_flags.kI3C_MasterAllErrorFlags (*C enumerator*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_error_flags.kI3C_MasterErrorCrcFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_error_flags.kI3C_MasterErrorInvalidReqFlag (*C enumerator*), 468

drivers_RT_RT700_MIMXRT798S._i3c_master_error_flags.kI3C_MasterErrorMsgFlag (*C enumerator*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_error_flags.kI3C_MasterErrorNackFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_error_flags.kI3C_MasterErrorParityFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_error_flags.kI3C_MasterErrorReadFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_error_flags.kI3C_MasterErrorTimeoutFlag (*C enumerator*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_error_flags.kI3C_MasterErrorWriteAbortFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_error_flags.kI3C_MasterErrorWriteFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_flags (*C enum*), 466
drivers_RT_RT700_MIMXRT798S._i3c_master_flags.kI3C_MasterArbitrationWonFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_flags.kI3C_MasterBetweenFlag (*C enumerator*), 466
drivers_RT_RT700_MIMXRT798S._i3c_master_flags.kI3C_MasterClearFlags (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_flags.kI3C_MasterCompleteFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_flags.kI3C_MasterControlDoneFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_flags.kI3C_MasterErrorFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_flags.kI3C_MasterNackDetectFlag (*C enumerator*), 466
drivers_RT_RT700_MIMXRT798S._i3c_master_flags.kI3C_MasterRxReadyFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_flags.kI3C_MasterSlave2MasterFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_flags.kI3C_MasterSlaveStartFlag (*C enumerator*), 466
drivers_RT_RT700_MIMXRT798S._i3c_master_flags.kI3C_MasterTxReadyFlag (*C enumerator*), 467
drivers_RT_RT700_MIMXRT798S._i3c_master_handle (*C struct*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_handle.callback (*C var*), 480
drivers_RT_RT700_MIMXRT798S._i3c_master_handle.ibiAddress (*C var*), 480
drivers_RT_RT700_MIMXRT798S._i3c_master_handle.ibiBuff (*C var*), 480
drivers_RT_RT700_MIMXRT798S._i3c_master_handle.ibiPayloadSize (*C var*), 480
drivers_RT_RT700_MIMXRT798S._i3c_master_handle.ibiType (*C var*), 480
drivers_RT_RT700_MIMXRT798S._i3c_master_handle.remainingBytes (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_handle.rxTermOps (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_handle.state (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_handle.transfer (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_handle.userData (*C var*), 480
drivers_RT_RT700_MIMXRT798S._i3c_master_hkeep (*C enum*), 469
drivers_RT_RT700_MIMXRT798S._i3c_master_hkeep.kI3C_MasterHighKeeperNone (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_master_hkeep.kI3C_MasterHighKeeperWiredIn (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_master_hkeep.kI3C_MasterPassiveSDA (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_master_hkeep.kI3C_MasterPassiveSDASCL (*C enumerator*), 469
drivers_RT_RT700_MIMXRT798S._i3c_master_src_clk (*C enum*), 473
drivers_RT_RT700_MIMXRT798S._i3c_master_src_clk.kI3C_MasterSrcClk24M (*C enumerator*), 473

drivers_RT_RT700_MIMXRT798S._i3c_master_src_clk.kI3C_MasterSrcClk25M (*C enumerator*), 473
drivers_RT_RT700_MIMXRT798S._i3c_master_state (*C enum*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_state.kI3C_MasterStateDaa (*C enumerator*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_state.kI3C_MasterStateDdr (*C enumerator*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_state.kI3C_MasterStateIbiAck (*C enumerator*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_state.kI3C_MasterStateIbiRcv (*C enumerator*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_state.kI3C_MasterStateIdle (*C enumerator*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_state.kI3C_MasterStateMsgSdr (*C enumerator*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_state.kI3C_MasterStateNormAct (*C enumerator*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_state.kI3C_MasterStateSlvReq (*C enumerator*), 468
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer (*C struct*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_callback (*C struct*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_callback.ibiCallback (*C var*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_callback.slave2Master (*C var*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_callback.transferComplete (*C var*), 478
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_flags (*C enum*), 473
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_flags.kI3C_TransferDefaultFlag (*C enumerator*), 473
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_flags.kI3C_TransferDisableRxTermFlag (*C enumerator*), 473
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_flags.kI3C_TransferNoStartFlag (*C enumerator*), 473
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_flags.kI3C_TransferNoStopFlag (*C enumerator*), 473
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_flags.kI3C_TransferRepeatedStartFlag (*C enumerator*), 473
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_flags.kI3C_TransferRxAutoTermFlag (*C enumerator*), 473
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_flags.kI3C_TransferStartWithBroadcastAddr (*C enumerator*), 473
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer_flags.kI3C_TransferWordsFlag (*C enumerator*), 473
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer.busType (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer.data (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer.dataSize (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer.direction (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer.flags (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer.ibiResponse (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer.slaveAddress (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer.subaddress (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_master_transfer.subaddressSize (*C var*), 479
drivers_RT_RT700_MIMXRT798S._i3c_register_ibi_addr (*C struct*), 475
drivers_RT_RT700_MIMXRT798S._i3c_register_ibi_addr.address (*C var*), 475
drivers_RT_RT700_MIMXRT798S._i3c_register_ibi_addr.i3cFastStart (*C var*), 475
drivers_RT_RT700_MIMXRT798S._i3c_register_ibi_addr.ibiHasPayload (*C var*), 476
drivers_RT_RT700_MIMXRT798S._i3c_rx_term_ops (*C enum*), 472
drivers_RT_RT700_MIMXRT798S._i3c_rx_term_ops.kI3C_RxAutoTerm (*C enumerator*), 472
drivers_RT_RT700_MIMXRT798S._i3c_rx_term_ops.kI3C_RxTermDisable (*C enumerator*), 472
drivers_RT_RT700_MIMXRT798S._i3c_rx_term_ops.kI3C_RxTermLastByte (*C enumerator*), 472
drivers_RT_RT700_MIMXRT798S._i3c_rx_trigger_level (*C enum*), 471
drivers_RT_RT700_MIMXRT798S._i3c_rx_trigger_level.kI3C_RxTriggerOnNotEmpty (*C enumerator*), 471
drivers_RT_RT700_MIMXRT798S._i3c_rx_trigger_level.kI3C_RxTriggerUntilOneHalfOrMore (*C*

enumerator), 471
drivers_RT_RT700_MIMXRT798S._i3c_rx_trigger_level.kI3C_RxTriggerUntilOneQuarterOrMore (*C enumerator*), 471
drivers_RT_RT700_MIMXRT798S._i3c_rx_trigger_level.kI3C_RxTriggerUntilThreeQuarterOrMore (*C enumerator*), 471
drivers_RT_RT700_MIMXRT798S._i3c_slave_activity_state (*C enum*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_activity_state.kI3C_SlaveLatency1Ms (*C enumerator*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_activity_state.kI3C_SlaveLatency10S (*C enumerator*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_activity_state.kI3C_SlaveLatency100Ms (*C enumerator*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_activity_state.kI3C_SlaveNoLatency (*C enumerator*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_config (*C struct*), 495
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.bcr (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.dcr (*C var*), 495
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.enableSlave (*C var*), 495
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.hdrMode (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.ignoreS0S1Error (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.matchSlaveStartStop (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.maxReadLength (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.maxWriteLength (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.nakAllRequest (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.offline (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.partNumber (*C var*), 495
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.staticAddr (*C var*), 495
drivers_RT_RT700_MIMXRT798S._i3c_slave_config.vendorID (*C var*), 495
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_handle (*C struct*), 499
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_handle.base (*C var*), 500
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_handle.callback (*C var*), 500
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_handle.eventMask (*C var*), 500
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_handle.isBusy (*C var*), 500
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_handle.isDdrMode (*C var*), 500
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_handle.rxDmaHandle (*C var*), 500
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_handle.transfer (*C var*), 500
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_handle.txDmaHandle (*C var*), 500
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_handle.userData (*C var*), 500
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_handle.wasTransmit (*C var*), 500
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_transfer (*C struct*), 499
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_transfer.completionStatus (*C var*), 499
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_transfer.event (*C var*), 499
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_transfer.rxData (*C var*), 499
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_transfer.rxDataSize (*C var*), 499
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_transfer.txData (*C var*), 499
drivers_RT_RT700_MIMXRT798S._i3c_slave_edma_transfer.txDataSize (*C var*), 499
drivers_RT_RT700_MIMXRT798S._i3c_slave_error_flags (*C enum*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_error_flags.kI3C_SlaveErrorHdrCRCFlag (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_error_flags.kI3C_SlaveErrorHdrParityFlag (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_error_flags.kI3C_SlaveErrorInvalidStartFlag (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_error_flags.kI3C_SlaveErrorOverreadFlag (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_error_flags.kI3C_SlaveErrorOverrunFlag (*C enumerator*), 492

drivers_RT_RT700_MIMXRT798S._i3c_slave_error_flags.kI3C_SlaveErrorOverwriteFlag (*C enumerator*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_error_flags.kI3C_SlaveErrorS0S1Flag (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_error_flags.kI3C_SlaveErrorSdrParityFlag (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_error_flags.kI3C_SlaveErrorTermFlag (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_error_flags.kI3C_SlaveErrorUnderrunFlag (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_error_flags.kI3C_SlaveErrorUnderrunNakFlag (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_event (*C enum*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_event.kI3C_SlaveEventHotJoinReq (*C enumerator*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_event.kI3C_SlaveEventIBI (*C enumerator*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_event.kI3C_SlaveEventMasterReq (*C enumerator*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_event.kI3C_SlaveEventNormal (*C enumerator*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags (*C enum*), 490
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveAllIrqFlags (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveBusDAAFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveBusHDRModeFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveBusStartFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveBusStopFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveCCCHandledFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveClearFlags (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveDynamicAddrChangedFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveErrorFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveEventSentFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveHDRCommandMatchFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveHotJoinDisabledFlag (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveIbiDisableFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveMasterRequestDisabledFlag (*C enumerator*), 492
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveMatchedFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveMessageFlag (*C enumerator*), 490
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveNotStopFlag (*C enumerator*), 490
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveReceivedCCCFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveRequiredReadFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveRequiredWriteFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveRxReadyFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_flags.kI3C_SlaveTxReadyFlag (*C enumerator*), 491
drivers_RT_RT700_MIMXRT798S._i3c_slave_handle (*C struct*), 497
drivers_RT_RT700_MIMXRT798S._i3c_slave_handle.callback (*C var*), 497
drivers_RT_RT700_MIMXRT798S._i3c_slave_handle.eventMask (*C var*), 497
drivers_RT_RT700_MIMXRT798S._i3c_slave_handle.isBusy (*C var*), 497
drivers_RT_RT700_MIMXRT798S._i3c_slave_handle.transfer (*C var*), 497
drivers_RT_RT700_MIMXRT798S._i3c_slave_handle.transferredCount (*C var*), 497
drivers_RT_RT700_MIMXRT798S._i3c_slave_handle.txFifoSize (*C var*), 497

drivers_RT_RT700_MIMXRT798S._i3c_slave_handle.userData (*C var*), 497
drivers_RT_RT700_MIMXRT798S._i3c_slave_handle.wasTransmit (*C var*), 497
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer (*C struct*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer_event (*C enum*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer_event.kI3C_SlaveAddressMatchEvent (*C enumerator*), 493
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer_event.kI3C_SlaveAllEvents (*C enumerator*), 494
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer_event.kI3C_SlaveCompletionEvent (*C enumerator*), 494
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer_event.kI3C_SlaveHDRCommandMatchEvent (*C enumerator*), 494
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer_event.kI3C_SlaveReceivedCCCEvent (*C enumerator*), 494
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer_event.kI3C_SlaveReceiveEvent (*C enumerator*), 494
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer_event.kI3C_SlaveRequestSentEvent (*C enumerator*), 494
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer_event.kI3C_SlaveRequiredTransmitEvent (*C enumerator*), 494
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer_event.kI3C_SlaveStartEvent (*C enumerator*), 494
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer_event.kI3C_SlaveTransmitEvent (*C enumerator*), 494
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer.completionStatus (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer.event (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer.rxData (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer.rxDataSize (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer.transferredCount (*C var*), 497
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer.txData (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_slave_transfer.txDataSize (*C var*), 496
drivers_RT_RT700_MIMXRT798S._i3c_start_scl_delay (*C enum*), 472
drivers_RT_RT700_MIMXRT798S._i3c_start_scl_delay.kI3C_IncreaseSclHalfPeriod (*C enumerator*), 472
drivers_RT_RT700_MIMXRT798S._i3c_start_scl_delay.kI3C_IncreaseSclOneAndHalfPeriod (*C enumerator*), 472
drivers_RT_RT700_MIMXRT798S._i3c_start_scl_delay.kI3C_IncreaseSclOnePeriod (*C enumerator*), 472
drivers_RT_RT700_MIMXRT798S._i3c_start_scl_delay.kI3C_NoDelay (*C enumerator*), 472
drivers_RT_RT700_MIMXRT798S._i3c_timing (*C struct*), 477
drivers_RT_RT700_MIMXRT798S._i3c_timing.actualI2cBaud (*C var*), 477
drivers_RT_RT700_MIMXRT798S._i3c_timing.actualOdBaud (*C var*), 477
drivers_RT_RT700_MIMXRT798S._i3c_timing.actualPpBaud (*C var*), 477
drivers_RT_RT700_MIMXRT798S._i3c_timing.i2cBaud (*C var*), 477
drivers_RT_RT700_MIMXRT798S._i3c_timing.odBaud (*C var*), 477
drivers_RT_RT700_MIMXRT798S._i3c_timing.odHighPP (*C var*), 477
drivers_RT_RT700_MIMXRT798S._i3c_timing.ppBaud (*C var*), 477
drivers_RT_RT700_MIMXRT798S._i3c_timing.ppLow (*C var*), 477
drivers_RT_RT700_MIMXRT798S._i3c_tx_trigger_level (*C enum*), 471
drivers_RT_RT700_MIMXRT798S._i3c_tx_trigger_level.kI3C_TxTriggerOnEmpty (*C enumerator*), 471
drivers_RT_RT700_MIMXRT798S._i3c_tx_trigger_level.kI3C_TxTriggerUntilOneHalfOrLess (*C enumerator*), 471
drivers_RT_RT700_MIMXRT798S._i3c_tx_trigger_level.kI3C_TxTriggerUntilOneLessThanFull (*C enumerator*), 471
drivers_RT_RT700_MIMXRT798S._i3c_tx_trigger_level.kI3C_TxTriggerUntilOneQuarterOrLess (*C enumerator*), 471
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t (*C enum*), 500

drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Adc0Tcomp0ToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Adc0Tcomp0ToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Adc0Tcomp0ToTimer6CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Adc0Tcomp0ToTimer6Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Adc0Tcomp0ToTimer7CaptureChannels (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Adc0Tcomp0ToTimer7Trigger (*C enumerator*), 512
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Adc0Tcomp1ToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Adc0Tcomp1ToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Adc0Tcomp1ToTimer6CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Adc0Tcomp1ToTimer6Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Adc0Tcomp1ToTimer7CaptureChannels (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Adc0Tcomp1ToTimer7Trigger (*C enumerator*), 512
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cmp0OutToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cmp0OutToFlexcomm17 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cmp0OutToFlexcomm18 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cmp0OutToFlexcomm19 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cmp0OutToFlexcomm20 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cmp0OutToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cmp0OutToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cmp0OutToTimer6CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cmp0OutToTimer6Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cmp0OutToTimer7CaptureChannels (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cmp0OutToTimer7Trigger (*C enumerator*), 512
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cpu0TxevToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Cpu1TxevToAdc0 (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct0Mat3ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct1Mat3ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct2Mat3ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct3Mat3ToAdc0 (*C enumerator*), 507

drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct4Mat3ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct5Mat3ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct5Mat3ToFlexcomm17 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct5Mat3ToFlexcomm18 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct5Mat3ToFlexcomm19 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct5Mat3ToFlexcomm20 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct6Mat3ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct6Mat3ToFlexcomm17 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct6Mat3ToFlexcomm18 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct6Mat3ToFlexcomm19 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct6Mat3ToFlexcomm20 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct7Mat3ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct7Mat3ToFlexcomm17 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct7Mat3ToFlexcomm18 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct7Mat3ToFlexcomm19 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ct7Mat3ToFlexcomm20 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ctimer5ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ctimer6ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Ctimer7ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp0ToFlexcomm17 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp0ToFlexcomm18 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp0ToFlexcomm19 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp0ToFlexcomm20 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp0ToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp0ToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp0ToTimer6CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp0ToTimer6Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp0ToTimer7CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp0ToTimer7Trigger (*C enumerator*), 511

drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp1ToFlexcomm17 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp1ToFlexcomm18 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp1ToFlexcomm19 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp1ToFlexcomm20 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp1ToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp1ToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp1ToTimer6CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp1ToTimer6Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp1ToTimer7CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp1ToTimer7Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp2ToFlexcomm17 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp2ToFlexcomm18 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp2ToFlexcomm19 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp2ToFlexcomm20 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp2ToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp2ToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp2ToTimer6CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp2ToTimer6Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp2ToTimer7CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp2ToTimer7Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp3ToFlexcomm17 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp3ToFlexcomm18 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp3ToFlexcomm19 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp3ToFlexcomm20 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp3ToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp3ToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp3ToTimer6CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp3ToTimer6Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp3ToTimer7CaptureChannels (*C enumerator*), 509

drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp3ToTimer7Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp4ToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp4ToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp4ToTimer6CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp4ToTimer6Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp4ToTimer7CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp4ToTimer7Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp5ToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp5ToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp5ToTimer6CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp5ToTimer6Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp5ToTimer7CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp5ToTimer7Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp6ToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp6ToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp6ToTimer6CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp6ToTimer6Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp6ToTimer7CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp6ToTimer7Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp7ToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp7ToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp7ToTimer6CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp7ToTimer6Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp7ToTimer7CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp7ToTimer7Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp8ToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp8ToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp8ToTimer6CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp8ToTimer6Trigger (*C enumerator*), 511

drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp8ToTimer7CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp8ToTimer7Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp9ToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp9ToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp9ToTimer6CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp9ToTimer6Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp9ToTimer7CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_CtInp9ToTimer7Trigger (*C enumerator*), 512
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma2Irq0ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma2Irq1ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma2Irq2ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma2Irq3ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma2Irq4ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma2Irq5ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma2Irq6ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma2Irq7ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma3Irq0ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma3Irq1ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma3Irq2ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma3Irq3ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma3Irq4ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma3Irq5ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma3Irq6ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Dma3Irq7ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_EzhvOutToAdc0 (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Flexcomm17ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Flexcomm18ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Flexcomm19ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Flexcomm20ToDspInterrupt (*C enumerator*), 503

drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio0Irq0ToAdc0 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio1Irq0ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio2Irq0ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio3Irq0ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio4Irq0ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio5Irq0ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio6Irq0ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio7Irq0ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio8Irq0ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio8Irq0ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio8Irq1ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio9Irq0ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio9Irq0ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio9Irq1ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio10Irq0ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio10Irq0ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Gpio10Irq1ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioInt0BmatchToAdc0 (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioInt0ToDspInterrupt (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioInt1BmatchToAdc0 (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioInt1BMatchToFlexcomm17 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioInt1BMatchToFlexcomm18 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioInt1BMatchToFlexcomm19 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioInt1BMatchToFlexcomm20 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioInt1ToDspInterrupt (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioInt2ToDspInterrupt (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioInt3ToDspInterrupt (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin0ToPintsel (*C enumerator*), 500
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin1ToPintsel (*C enumerator*), 500

drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin2ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin3ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin4ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin5ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin6ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin7ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin8ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin9ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin10ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin11ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin12ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin13ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin14ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin15ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin16ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin17ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin18ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin19ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin20ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin21ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin22ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin23ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin24ToPintsel (*C enumerator*), 501
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin25ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin26ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin27ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin28ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin29ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin30ToPintsel (*C enumerator*), 502

drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort8Pin31ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin0ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin1ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin2ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin3ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin4ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin5ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin6ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin7ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin8ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin9ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin10ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin11ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin12ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin13ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin14ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin15ToPintsel (*C enumerator*), 502
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin16ToPintsel (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_GpioPort10Pin17ToPintsel (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_HwvadToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_I3c2ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_I3c3ToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_MicfilToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Mrt1ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Mu0BToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Mu3BToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_OsEventTimerToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Pmc1ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Rtc1AlarmToDspInterrupt (*C enumerator*), 504

drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Rtc1IrqToFlexcomm17 (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Rtc1IrqToFlexcomm18 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Rtc1IrqToFlexcomm19 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Rtc1IrqToFlexcomm20 (*C enumerator*), 506
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Rtc1WakeupToDspInterrupt (*C enumerator*), 504
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3RxSyncToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3RxSyncToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3RxSyncToTimer6CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3RxSyncToTimer6Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3RxSyncToTimer7CaptureChannels (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3RxSyncToTimer7Trigger (*C enumerator*), 512
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3ToDspInterrupt (*C enumerator*), 505
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3TxSyncToTimer5CaptureChannels (*C enumerator*), 508
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3TxSyncToTimer5Trigger (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3TxSyncToTimer6CaptureChannels (*C enumerator*), 509
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3TxSyncToTimer6Trigger (*C enumerator*), 511
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3TxSyncToTimer7CaptureChannels (*C enumerator*), 510
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sai3TxSyncToTimer7Trigger (*C enumerator*), 512
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sct0Out4ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sct0Out5ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Sct0Out9ToAdc0 (*C enumerator*), 507
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Utick1ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Wdt2ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._inputmux_connection_t.kINPUTMUX_Wdt3ToDspInterrupt (*C enumerator*), 503
drivers_RT_RT700_MIMXRT798S._iopctl_group (*C struct*), 516
drivers_RT_RT700_MIMXRT798S._irtc_alarm_match (*C enum*), 522
drivers_RT_RT700_MIMXRT798S._irtc_alarm_match.kRTC_MatchSecMinHr (*C enumerator*), 522
drivers_RT_RT700_MIMXRT798S._irtc_alarm_match.kRTC_MatchSecMinHrDay (*C enumerator*), 522
drivers_RT_RT700_MIMXRT798S._irtc_alarm_match.kRTC_MatchSecMinHrDayMnth (*C enumerator*), 522
drivers_RT_RT700_MIMXRT798S._irtc_alarm_match.kRTC_MatchSecMinHrDayMnthYr (*C enumerator*), 522
drivers_RT_RT700_MIMXRT798S._irtc_clock_select (*C enum*), 519

drivers_RT_RT700_MIMXRT798S._irtc_clock_select.kIRTC_Clk16K (*C enumerator*), 519
drivers_RT_RT700_MIMXRT798S._irtc_clock_select.kIRTC_Clk32K (*C enumerator*), 519
drivers_RT_RT700_MIMXRT798S._irtc_clockout_sel (*C enum*), 522
drivers_RT_RT700_MIMXRT798S._irtc_clockout_sel.kIRTC_Clkout32kHz (*C enumerator*), 522
drivers_RT_RT700_MIMXRT798S._irtc_clockout_sel.kIRTC_ClkoutCoarse1Hz (*C enumerator*), 522
drivers_RT_RT700_MIMXRT798S._irtc_clockout_sel.kIRTC_ClkoutFine1Hz (*C enumerator*), 522
drivers_RT_RT700_MIMXRT798S._irtc_clockout_sel.kIRTC_ClkoutNo (*C enumerator*), 522
drivers_RT_RT700_MIMXRT798S._irtc_config (*C struct*), 525
drivers_RT_RT700_MIMXRT798S._irtc_config.alrmMatch (*C var*), 525
drivers_RT_RT700_MIMXRT798S._irtc_config.clockSelect (*C var*), 525
drivers_RT_RT700_MIMXRT798S._irtc_config.disableClockOutput (*C var*), 525
drivers_RT_RT700_MIMXRT798S._irtc_datetime (*C struct*), 524
drivers_RT_RT700_MIMXRT798S._irtc_datetime.day (*C var*), 524
drivers_RT_RT700_MIMXRT798S._irtc_datetime.hour (*C var*), 524
drivers_RT_RT700_MIMXRT798S._irtc_datetime.minute (*C var*), 524
drivers_RT_RT700_MIMXRT798S._irtc_datetime.month (*C var*), 524
drivers_RT_RT700_MIMXRT798S._irtc_datetime.second (*C var*), 524
drivers_RT_RT700_MIMXRT798S._irtc_datetime.weekDay (*C var*), 524
drivers_RT_RT700_MIMXRT798S._irtc_datetime.year (*C var*), 524
drivers_RT_RT700_MIMXRT798S._irtc_daylight_time (*C struct*), 524
drivers_RT_RT700_MIMXRT798S._irtc_daylight_time.endDay (*C var*), 525
drivers_RT_RT700_MIMXRT798S._irtc_daylight_time.endHour (*C var*), 525
drivers_RT_RT700_MIMXRT798S._irtc_daylight_time.endMonth (*C var*), 525
drivers_RT_RT700_MIMXRT798S._irtc_daylight_time.startDay (*C var*), 525
drivers_RT_RT700_MIMXRT798S._irtc_daylight_time.startHour (*C var*), 525
drivers_RT_RT700_MIMXRT798S._irtc_daylight_time.startMonth (*C var*), 525
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable (*C enum*), 519
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_1hzInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_2hzInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_4hzInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_8hzInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_16hzInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_32hzInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_64hzInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_128hzInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_256hzInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_512hzInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_AlarmInterruptEnable (*C enumerator*), 519
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_DayInterruptEnable (*C enumerator*), 519
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_HourInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_MinInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_TamperQueueFullInterruptEnable (*C enumerator*), 520

drivers_RT_RT700_MIMXRT798S._irtc_interrupt_enable.kIRTC_WakeTimerInterruptEnable (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_status_flags (*C enum*), 520
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_1hzFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_2hzFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_4hzFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_8hzFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_16hzFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_32hzFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_64hzFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_128hzFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_256hzFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_512hzFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_AlarmFlag (*C enumerator*), 520
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_BusErrFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_DayFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_HourFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_InvalidFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_MinFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_WakeTimerFlag (*C enumerator*), 522
drivers_RT_RT700_MIMXRT798S._irtc_status_flags.kIRTC_WriteProtFlag (*C enumerator*), 521
drivers_RT_RT700_MIMXRT798S._itrc_enable (*C enum*), 528
drivers_RT_RT700_MIMXRT798S._itrc_enable.kITRC_Disable (*C enumerator*), 528
drivers_RT_RT700_MIMXRT798S._itrc_enable.kITRC_Enable (*C enumerator*), 528
drivers_RT_RT700_MIMXRT798S._itrc_input_signals (*C enum*), 528
drivers_RT_RT700_MIMXRT798S._itrc_lock (*C enum*), 528
drivers_RT_RT700_MIMXRT798S._itrc_lock.kITRC_Lock (*C enumerator*), 528
drivers_RT_RT700_MIMXRT798S._itrc_lock.kITRC_Unlock (*C enumerator*), 528
drivers_RT_RT700_MIMXRT798S._itrc_out_signals (*C enum*), 528
drivers_RT_RT700_MIMXRT798S._itrc_out_signals.kITRC_Out_0 (*C enumerator*), 528
drivers_RT_RT700_MIMXRT798S._itrc_out_signals.kITRC_Out_1 (*C enumerator*), 528
drivers_RT_RT700_MIMXRT798S._itrc_out_signals.kITRC_Out_2 (*C enumerator*), 528
drivers_RT_RT700_MIMXRT798S._itrc_out_signals.kITRC_Out_3 (*C enumerator*), 528
drivers_RT_RT700_MIMXRT798S._itrc_out_signals.kITRC_Out_4 (*C enumerator*), 528
drivers_RT_RT700_MIMXRT798S._itrc_out_signals.kITRC_Out_5 (*C enumerator*), 528
drivers_RT_RT700_MIMXRT798S._itrc_out_signals.kITRC_Out_6 (*C enumerator*), 528
drivers_RT_RT700_MIMXRT798S._jpegdec_config (*C struct*), 538
drivers_RT_RT700_MIMXRT798S._jpegdec_decoder_config (*C struct*), 538
drivers_RT_RT700_MIMXRT798S._jpegdec_descpt (*C struct*), 538
drivers_RT_RT700_MIMXRT798S._jpegdec_endian_mode (*C enum*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_endian_mode.kJPEGDEC_BigEndian (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_endian_mode.kJPEGDEC_LittleEndian (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_flags (*C enum*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_flags.kJPEGDEC_AllFlags (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_flags.kJPEGDEC_BitReadErrorFlag (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_flags.kJPEGDEC_DecodeCompleteFlag (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_flags.kJPEGDEC_DecodeErrorFlag (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_flags.kJPEGDEC_DescptReadErrorFlag (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_flags.kJPEGDEC_ErrorFlags (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_flags.kJPEGDEC_PixelWriteErrorFlag (*C enumerator*), 531

drivers_RT_RT700_MIMXRT798S._jpegdec_flags.kJPEGDEC_StreamBufferDoneFlag (*C enumerator*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_flags.kJPEGDEC_StreamBufferHalfDoneFlag (*C enumerator*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_flags.kJPEGDEC_SwitchInFlag (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_pixel_depth (*C enum*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_pixel_depth.kJPEGDEC_PixelDepth8Bit (*C enumerator*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_pixel_depth.kJPEGDEC_PixelDepth12Bit (*C enumerator*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_pixel_format (*C enum*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_pixel_format.kJPEGDEC_PixelFormatGray (*C enumerator*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_pixel_format.kJPEGDEC_PixelFormatReserved (*C enumerator*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_pixel_format.kJPEGDEC_PixelFormatRGB (*C enumerator*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_pixel_format.kJPEGDEC_PixelFormatYCCK (*C enumerator*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_pixel_format.kJPEGDEC_PixelFormatYUV420 (*C enumerator*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_pixel_format.kJPEGDEC_PixelFormatYUV422 (*C enumerator*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_pixel_format.kJPEGDEC_PixelFormatYUV444 (*C enumerator*), 530
drivers_RT_RT700_MIMXRT798S._jpegdec_slots (*C enum*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_slots.kJPEGDEC_Slot0 (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_slots.kJPEGDEC_Slot1 (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_slots.kJPEGDEC_Slot2 (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._jpegdec_slots.kJPEGDEC_Slot3 (*C enumerator*), 531
drivers_RT_RT700_MIMXRT798S._lcdif_cursor_config (*C struct*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_cursor_config.enable (*C var*), 581
drivers_RT_RT700_MIMXRT798S._lcdif_cursor_config.format (*C var*), 581
drivers_RT_RT700_MIMXRT798S._lcdif_cursor_config.hotspotOffsetX (*C var*), 581
drivers_RT_RT700_MIMXRT798S._lcdif_cursor_config.hotspotOffsetY (*C var*), 581
drivers_RT_RT700_MIMXRT798S._lcdif_cursor_format (*C enum*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_cursor_format.kLCDIF_CursorARGB8888 (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_cursor_format.kLCDIF_CursorMasked (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_cmd_flag (*C enum*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_cmd_flag.kLCDIF_DbiCmdAddress (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_cmd_flag.kLCDIF_DbiCmdData (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_cmd_flag.kLCDIF_DbiCmdReadMem (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_cmd_flag.kLCDIF_DbiCmdWriteMem (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config (*C struct*), 581
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.acTimeUnit (*C var*), 582
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.format (*C var*), 582
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.reversePolarity (*C var*), 582
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.swizzle (*C var*), 582
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.type (*C var*), 582
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.typeCSCLTwrh (*C var*), 582
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.typeCSCLTwrl (*C var*), 582
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.typeCTas (*C var*), 582

drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.writeCSAssert (*C var*), 582
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.writeCSDeassert (*C var*), 582
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.writeWRAssert (*C var*), 582
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.writeWRDeassert (*C var*), 582
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_config.writeWRPeriod (*C var*), 582
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format (*C enum*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD1RG888Option1 (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD1RG888Option2 (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD1RG888Option3 (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD1RGB565Option1 (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD1RGB565Option2 (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD1RGB565Option3 (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD8RGB332 (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD8RGB444 (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD8RGB565 (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD8RGB666 (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD8RGB888 (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD9RGB666 (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD16RGB332 (*C enumerator*), 573
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD16RGB444 (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD16RGB565 (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD16RGB666Option1 (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD16RGB666Option2 (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD16RGB888Option1 (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_format.kLCDIF_DbiOutD16RGB888Option2 (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_swizzle (*C enum*), 575
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_swizzle.kLCDIF_DbiOutSwizzleBGR (*C enumerator*), 575
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_out_swizzle.kLCDIF_DbiOutSwizzleRGB (*C enumerator*), 575
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_type (*C enum*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_type.kLCDIF_DbiTypeA_ClockedE (*C enumerator*), 575
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_type.kLCDIF_DbiTypeA_FixedE (*C enumerator*), 574
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_type.kLCDIF_DbiTypeB (*C enumerator*), 575
drivers_RT_RT700_MIMXRT798S._lcdif_dbi_type.kLCDIF_DbiTypeC (*C enumerator*), 575
drivers_RT_RT700_MIMXRT798S._lcdif_dither_config (*C struct*), 581
drivers_RT_RT700_MIMXRT798S._lcdif_dither_config.enable (*C var*), 581

drivers_RT_RT700_MIMXRT798S._lcdif_dither_config.high (*C var*), 581
drivers_RT_RT700_MIMXRT798S._lcdif_dither_config.low (*C var*), 581
drivers_RT_RT700_MIMXRT798S._lcdif_dpi_config (*C struct*), 577
drivers_RT_RT700_MIMXRT798S._lcdif_dpi_config.format (*C var*), 578
drivers_RT_RT700_MIMXRT798S._lcdif_dpi_config.hbp (*C var*), 577
drivers_RT_RT700_MIMXRT798S._lcdif_dpi_config.hfp (*C var*), 577
drivers_RT_RT700_MIMXRT798S._lcdif_dpi_config.hsw (*C var*), 577
drivers_RT_RT700_MIMXRT798S._lcdif_dpi_config.panelHeight (*C var*), 577
drivers_RT_RT700_MIMXRT798S._lcdif_dpi_config.panelWidth (*C var*), 577
drivers_RT_RT700_MIMXRT798S._lcdif_dpi_config.polarityFlags (*C var*), 578
drivers_RT_RT700_MIMXRT798S._lcdif_dpi_config.vbp (*C var*), 578
drivers_RT_RT700_MIMXRT798S._lcdif_dpi_config.vfp (*C var*), 578
drivers_RT_RT700_MIMXRT798S._lcdif_dpi_config.vsw (*C var*), 577
drivers_RT_RT700_MIMXRT798S._lcdif_endian_mode (*C enum*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_endian_mode.kLCDIF_HalfWordSwap (*C enumerator*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_endian_mode.kLCDIF_NoSwap (*C enumerator*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_endian_mode.kLCDIF_WordSwap (*C enumerator*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config (*C struct*), 579
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.alpha (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.clearValue (*C var*), 579
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.colorkey (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.decompress (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.enable (*C var*), 579
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.enableClear (*C var*), 579
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.enableUVSwizzle (*C var*), 579
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.format (*C var*), 579
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.height (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.inOrder (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.rotateFlipMode (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.standard (*C var*), 579
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.topLeftX (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.topLeftY (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_fb_config.width (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_fb_format (*C enum*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_fb_format.kLCDIF_PixelFormatARGB1555 (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_fb_format.kLCDIF_PixelFormatARGB4444 (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_fb_format.kLCDIF_PixelFormatARGB8565 (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_fb_format.kLCDIF_PixelFormatARGB8565Tiled (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_fb_format.kLCDIF_PixelFormatARGB8888 (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_fb_format.kLCDIF_PixelFormatARGB8888Tiled (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_fb_format.kLCDIF_PixelFormatRGB565 (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_fb_format.kLCDIF_PixelFormatRGB888 (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_fb_format.kLCDIF_PixelFormatRGB888Tiled (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_fb_format.kLCDIF_PixelFormatYUV420Tiled (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_fb_format.kLCDIF_PixelFormatYUV422Tiled (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_interrupt (*C enum*), 572

drivers_RT_RT700_MIMXRT798S._lcdif_interrupt.kLCDIF_BusErrorInterrupt (*C enumerator*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_interrupt.kLCDIF_Display0DbiConfigErrorInterrupt (*C enumerator*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_interrupt.kLCDIF_Display0FrameDoneInterrupt (*C enumerator*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_interrupt.kLCDIF_PanelUnderflowInterrupt (*C enumerator*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_interrupt.kLCDIF_SoftwareResetDoneInterrupt (*C enumerator*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_layer_colorkey (*C struct*), 578
drivers_RT_RT700_MIMXRT798S._lcdif_layer_colorkey.enable (*C var*), 578
drivers_RT_RT700_MIMXRT798S._lcdif_layer_convert_standard (*C enum*), 571
drivers_RT_RT700_MIMXRT798S._lcdif_layer_convert_standard.kLCDIF_ConvertBT601 (*C enumerator*), 571
drivers_RT_RT700_MIMXRT798S._lcdif_layer_convert_standard.kLCDIF_ConvertBT709 (*C enumerator*), 571
drivers_RT_RT700_MIMXRT798S._lcdif_layer_decompress_mode (*C enum*), 569
drivers_RT_RT700_MIMXRT798S._lcdif_layer_decompress_mode.kLCDIF_DecompressDECNanoHSample (*C enumerator*), 569
drivers_RT_RT700_MIMXRT798S._lcdif_layer_decompress_mode.kLCDIF_DecompressDECNanoHVSample (*C enumerator*), 569
drivers_RT_RT700_MIMXRT798S._lcdif_layer_decompress_mode.kLCDIF_DecompressDECNanoNoneSample (*C enumerator*), 569
drivers_RT_RT700_MIMXRT798S._lcdif_layer_decompress_mode.kLCDIF_DecompressETC2 (*C enumerator*), 569
drivers_RT_RT700_MIMXRT798S._lcdif_layer_decompress_mode.kLCDIF_DecompressNone (*C enumerator*), 569
drivers_RT_RT700_MIMXRT798S._lcdif_layer_input_order (*C enum*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_layer_input_order.kLCDIF_PixelInputOrderABGR (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_layer_input_order.kLCDIF_PixelInputOrderARGB (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_layer_input_order.kLCDIF_PixelInputOrderBGRA (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_layer_input_order.kLCDIF_PixelInputOrderRGBA (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_layer_order (*C enum*), 571
drivers_RT_RT700_MIMXRT798S._lcdif_layer_order.kLCDIF_Overlay0Overlay1Video (*C enumerator*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_layer_order.kLCDIF_Overlay0VideoOverlay1 (*C enumerator*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_layer_order.kLCDIF_Overlay1Overlay0Video (*C enumerator*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_layer_order.kLCDIF_Overlay1VideoOverlay0 (*C enumerator*), 572
drivers_RT_RT700_MIMXRT798S._lcdif_layer_order.kLCDIF_VideoOverlay0Overlay1 (*C enumerator*), 571
drivers_RT_RT700_MIMXRT798S._lcdif_layer_order.kLCDIF_VideoOverlay1Overlay0 (*C enumerator*), 571
drivers_RT_RT700_MIMXRT798S._lcdif_layer_rotate_flip (*C enum*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_layer_rotate_flip.kLCDIF_FlipHorizontal (*C enumerator*), 569
drivers_RT_RT700_MIMXRT798S._lcdif_layer_rotate_flip.kLCDIF_FlipVertical (*C enumerator*), 569
drivers_RT_RT700_MIMXRT798S._lcdif_layer_rotate_flip.kLCDIF_Rotate0 (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_layer_rotate_flip.kLCDIF_Rotate90 (*C enumerator*), 568
drivers_RT_RT700_MIMXRT798S._lcdif_layer_rotate_flip.kLCDIF_Rotate180 (*C enumerator*),

569
drivers_RT_RT700_MIMXRT798S._lcdif_layer_rotate_flip.kLCDIF_Rotate270 (*C enumerator*), 569
drivers_RT_RT700_MIMXRT798S._lcdif_output_format (*C enum*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_output_format.kLCDIF_Output16BitConfig1 (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_output_format.kLCDIF_Output16BitConfig2 (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_output_format.kLCDIF_Output16BitConfig3 (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_output_format.kLCDIF_Output18BitConfig1 (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_output_format.kLCDIF_Output18BitConfig2 (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_output_format.kLCDIF_Output24Bit (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_panel_config (*C struct*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_panel_config.enable (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_panel_config.enableGamma (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_panel_config.endian (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_panel_config.order (*C var*), 580
drivers_RT_RT700_MIMXRT798S._lcdif_polarity_flags (*C enum*), 566
drivers_RT_RT700_MIMXRT798S._lcdif_polarity_flags.kLCDIF_DataEnableActiveHigh (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_polarity_flags.kLCDIF_DataEnableActiveLow (*C enumerator*), 566
drivers_RT_RT700_MIMXRT798S._lcdif_polarity_flags.kLCDIF_DriveDataOnFallingClkEdge (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_polarity_flags.kLCDIF_DriveDataOnRisingClkEdge (*C enumerator*), 567
drivers_RT_RT700_MIMXRT798S._lcdif_polarity_flags.kLCDIF_HsyncActiveHigh (*C enumerator*), 566
drivers_RT_RT700_MIMXRT798S._lcdif_polarity_flags.kLCDIF_HsyncActiveLow (*C enumerator*), 566
drivers_RT_RT700_MIMXRT798S._lcdif_polarity_flags.kLCDIF_VsyncActiveHigh (*C enumerator*), 566
drivers_RT_RT700_MIMXRT798S._lcdif_polarity_flags.kLCDIF_VsyncActiveLow (*C enumerator*), 566
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode (*C enum*), 570
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffAtop (*C enumerator*), 570
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffClear (*C enumerator*), 571
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffDst (*C enumerator*), 570
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffDstAtop (*C enumerator*), 570
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffDstIn (*C enumerator*), 571
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffDstOut (*C enumerator*), 571
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffDstOver (*C enumerator*), 571
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffIn (*C enumerator*), 570
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffMax (*C enumerator*), 571
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffOut (*C enu-*

merator), 570
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffOver (*C enumerator*), 570
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffPlus (*C enumerator*), 571
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffSrc (*C enumerator*), 570
drivers_RT_RT700_MIMXRT798S._lcdif_porter_duff_blend_mode.kLCDIF_PorterDuffXor (*C enumerator*), 571
drivers_RT_RT700_MIMXRT798S._ldr_buf (*C struct*), 882
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3 (*C struct*), 883
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3.Action (*C var*), 883
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3.block_buffer_size (*C var*), 883
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3.block_data_size (*C var*), 883
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3.block_data_total (*C var*), 883
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3.block_size (*C var*), 883
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3.commandSet (*C var*), 884
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3.data_buffer (*C var*), 884
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3.fromAPI (*C var*), 884
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3.in_data_block (*C var*), 883
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3.in_data_range (*C var*), 884
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3.in_data_section (*C var*), 884
drivers_RT_RT700_MIMXRT798S._ldr_Context_v3.processedBlocks (*C var*), 883
drivers_RT_RT700_MIMXRT798S._loader_command_sb3 (*C enum*), 879
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdCall (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdConfigMem (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdCopy (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdErase (*C enumerator*), 879
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdExecute (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdFillMem (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdFwVerCheck (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdInvalid (*C enumerator*), 879
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdLoad (*C enumerator*), 879
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdLoadCmac (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdLoadHashLocking (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdLoadKeyBlob (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdProgramFuse (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S._loader_command_sb3.kSB3_CmdProgramIFR (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S._lpadc_calibration_value (*C struct*), 604
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_average_mode (*C enum*), 587
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_average_mode.kLPADC_ConversionAverage1 (*C enumerator*), 587
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_average_mode.kLPADC_ConversionAverage2 (*C enumerator*), 587
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_average_mode.kLPADC_ConversionAverage4 (*C enumerator*), 587
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_average_mode.kLPADC_ConversionAverage8 (*C enumerator*), 587
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_average_mode.kLPADC_ConversionAverage16 (*C enumerator*), 587
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_average_mode.kLPADC_ConversionAverage32

(*C enumerator*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_average_mode.kLPADC_ConversionAverage64 (*C enumerator*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_average_mode.kLPADC_ConversionAverage128 (*C enumerator*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_average_mode.kLPADC_ConversionAverageMax (*C enumerator*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_resolution_mode (*C enum*), 587
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_resolution_mode.kLPADC_ConversionResolutionHigh (*C enumerator*), 587
drivers_RT_RT700_MIMXRT798S._lpadc_conversion_resolution_mode.kLPADC_ConversionResolutionStandard (*C enumerator*), 587
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_average_mode (*C enum*), 585
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount1 (*C enumerator*), 585
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount2 (*C enumerator*), 585
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount4 (*C enumerator*), 585
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount8 (*C enumerator*), 585
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount16 (*C enumerator*), 585
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount32 (*C enumerator*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount64 (*C enumerator*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount128 (*C enumerator*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_compare_mode (*C enum*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_compare_mode.kLPADC_HardwareCompareDisabled (*C enumerator*), 587
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_compare_mode.kLPADC_HardwareCompareRepeatUntilTrue (*C enumerator*), 587
drivers_RT_RT700_MIMXRT798S._lpadc_hardware_compare_mode.kLPADC_HardwareCompareStoreOnTrue (*C enumerator*), 587
drivers_RT_RT700_MIMXRT798S._lpadc_interrupt_enable (*C enum*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_interrupt_enable.kLPADC_FIFO0WatermarkInterruptEnable (*C enumerator*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_interrupt_enable.kLPADC_FIFOWatermarkInterruptEnable (*C enumerator*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_interrupt_enable.kLPADC_ResultFIFO0OverflowInterruptEnable (*C enumerator*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_interrupt_enable.kLPADC_ResultFIFOOverflowInterruptEnable (*C enumerator*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_interrupt_enable.kLPADC_Trigger0CompletionInterruptEnable (*C enumerator*), 584
drivers_RT_RT700_MIMXRT798S._lpadc_interrupt_enable.kLPADC_Trigger1CompletionInterruptEnable (*C enumerator*), 584
drivers_RT_RT700_MIMXRT798S._lpadc_interrupt_enable.kLPADC_TriggerExceptionInterruptEnable (*C enumerator*), 584
drivers_RT_RT700_MIMXRT798S._lpadc_offset_calibration_mode (*C enum*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_offset_calibration_mode.kLPADC_OffsetCalibration12bitMode (*C enumerator*), 589
drivers_RT_RT700_MIMXRT798S._lpadc_offset_calibration_mode.kLPADC_OffsetCalibration16bitMode (*C enumerator*), 589
drivers_RT_RT700_MIMXRT798S._lpadc_power_level_mode (*C enum*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_power_level_mode.kLPADC_PowerLevelAlt1 (*C enu-*

merator), 588
drivers_RT_RT700_MIMXRT798S._lpadc_power_level_mode.kLPADC_PowerLevelAlt2 (*C enumerator*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_power_level_mode.kLPADC_PowerLevelAlt3 (*C enumerator*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_power_level_mode.kLPADC_PowerLevelAlt4 (*C enumerator*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_reference_voltage_mode (*C enum*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_reference_voltage_mode.kLPADC_ReferenceVoltageAlt1 (*C enumerator*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_reference_voltage_mode.kLPADC_ReferenceVoltageAlt2 (*C enumerator*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_reference_voltage_mode.kLPADC_ReferenceVoltageAlt3 (*C enumerator*), 588
drivers_RT_RT700_MIMXRT798S._lpadc_sample_channel_mode (*C enum*), 584
drivers_RT_RT700_MIMXRT798S._lpadc_sample_channel_mode.kLPADC_SampleChannelDiffBothSide (*C enumerator*), 585
drivers_RT_RT700_MIMXRT798S._lpadc_sample_channel_mode.kLPADC_SampleChannelDiffBothSideAB (*C enumerator*), 585
drivers_RT_RT700_MIMXRT798S._lpadc_sample_channel_mode.kLPADC_SampleChannelDiffBothSideBA (*C enumerator*), 585
drivers_RT_RT700_MIMXRT798S._lpadc_sample_channel_mode.kLPADC_SampleChannelDualSingleEndBothSide (*C enumerator*), 585
drivers_RT_RT700_MIMXRT798S._lpadc_sample_channel_mode.kLPADC_SampleChannelSingleEndSideA (*C enumerator*), 585
drivers_RT_RT700_MIMXRT798S._lpadc_sample_channel_mode.kLPADC_SampleChannelSingleEndSideB (*C enumerator*), 585
drivers_RT_RT700_MIMXRT798S._lpadc_sample_scale_mode (*C enum*), 584
drivers_RT_RT700_MIMXRT798S._lpadc_sample_scale_mode.kLPADC_SampleFullScale (*C enumerator*), 584
drivers_RT_RT700_MIMXRT798S._lpadc_sample_scale_mode.kLPADC_SamplePartScale (*C enumerator*), 584
drivers_RT_RT700_MIMXRT798S._lpadc_sample_time_mode (*C enum*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK3 (*C enumerator*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK5 (*C enumerator*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK7 (*C enumerator*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK11 (*C enumerator*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK19 (*C enumerator*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK35 (*C enumerator*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK67 (*C enumerator*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK131 (*C enumerator*), 586
drivers_RT_RT700_MIMXRT798S._lpadc_status_flags (*C enum*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_status_flags.kLPADC_ActiveFlag (*C enumerator*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_status_flags.kLPADC_CalibrationReadyFlag (*C enumerator*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_status_flags.kLPADC_ResultFIFO0OverflowFlag (*C enumerator*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_status_flags.kLPADC_ResultFIFO0ReadyFlag (*C enumerator*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_status_flags.kLPADC_ResultFIFOOverflowFlag (*C enu-*

merator), 583
drivers_RT_RT700_MIMXRT798S._lpadc_status_flags.kLPADC_ResultFIFOReadyFlag (*C enumerator*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_status_flags.kLPADC_TriggerCompletionFlag (*C enumerator*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_status_flags.kLPADC_TriggerExceptionFlag (*C enumerator*), 583
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_priority_policy (*C enum*), 589
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_priority_policy.kLPADC_ConvPreemptImmediatelyAutoRestarte (*C enumerator*), 589
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_priority_policy.kLPADC_ConvPreemptImmediatelyAutoResumed (*C enumerator*), 589
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_priority_policy.kLPADC_ConvPreemptImmediatelyNotAutoResu (*C enumerator*), 589
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_priority_policy.kLPADC_ConvPreemptSoftlyAutoRestarted (*C enumerator*), 589
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_priority_policy.kLPADC_ConvPreemptSoftlyAutoResumed (*C enumerator*), 589
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_priority_policy.kLPADC_ConvPreemptSoftlyNotAutoResumed (*C enumerator*), 589
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_priority_policy.kLPADC_TriggerPriorityExceptionDisabled (*C enumerator*), 590
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_priority_policy.kLPADC_TriggerPriorityPreemptImmediately (*C enumerator*), 590
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_priority_policy.kLPADC_TriggerPriorityPreemptSoftly (*C enumerator*), 590
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_status_flags (*C enum*), 584
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_status_flags.kLPADC_Trigger0CompletedFlag (*C enumerator*), 584
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_status_flags.kLPADC_Trigger0InterruptedFlag (*C enumerator*), 584
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_status_flags.kLPADC_Trigger1CompletedFlag (*C enumerator*), 584
drivers_RT_RT700_MIMXRT798S._lpadc_trigger_status_flags.kLPADC_Trigger1InterruptedFlag (*C enumerator*), 584
drivers_RT_RT700_MIMXRT798S._lpadc_tune_value (*C enum*), 590
drivers_RT_RT700_MIMXRT798S._lpadc_tune_value.kLPADC_TuneValue0 (*C enumerator*), 590
drivers_RT_RT700_MIMXRT798S._lpadc_tune_value.kLPADC_TuneValue1 (*C enumerator*), 590
drivers_RT_RT700_MIMXRT798S._lpadc_tune_value.kLPADC_TuneValue2 (*C enumerator*), 590
drivers_RT_RT700_MIMXRT798S._lpadc_tune_value.kLPADC_TuneValue3 (*C enumerator*), 590
drivers_RT_RT700_MIMXRT798S._lpflexcomm_interrupt_flag (*C enum*), 611
drivers_RT_RT700_MIMXRT798S._lpflexcomm_interrupt_flag.kLPFLEXCOMM_AllInterruptFlag (*C enumerator*), 611
drivers_RT_RT700_MIMXRT798S._lpflexcomm_interrupt_flag.kLPFLEXCOMM_I2cMasterInterruptFlag (*C enumerator*), 611
drivers_RT_RT700_MIMXRT798S._lpflexcomm_interrupt_flag.kLPFLEXCOMM_I2cSlaveInterruptFlag (*C enumerator*), 611
drivers_RT_RT700_MIMXRT798S._lpflexcomm_interrupt_flag.kLPFLEXCOMM_SpiInterruptFlag (*C enumerator*), 611
drivers_RT_RT700_MIMXRT798S._lpflexcomm_interrupt_flag.kLPFLEXCOMM_UartRxInterruptFlag (*C enumerator*), 611
drivers_RT_RT700_MIMXRT798S._lpflexcomm_interrupt_flag.kLPFLEXCOMM_UartTxInterruptFlag (*C enumerator*), 611
drivers_RT_RT700_MIMXRT798S._lpi2c_data_match_config_mode (*C enum*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_data_match_config_mode.kLPI2C_1stWordAndM1EqualsM0AndM1 (*C enumerator*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_data_match_config_mode.kLPI2C_1stWordEqualsM0And2ndWordEquals (*C enumerator*), 623

drivers_RT_RT700_MIMXRT798S._lpi2c_data_match_config_mode.kLPI2C_1stWordEqualsM0OrM1 (*C enumerator*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_data_match_config_mode.kLPI2C_AnyWordAndM1EqualsM0AndM1 (*C enumerator*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_data_match_config_mode.kLPI2C_AnyWordEqualsM0AndNextWordEqu (*C enumerator*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_data_match_config_mode.kLPI2C_AnyWordEqualsM0OrM1 (*C enumerator*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_data_match_config_mode.kLPI2C_MatchDisabled (*C enumerator*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_direction (*C enum*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_direction.kLPI2C_Read (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_direction.kLPI2C_Write (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_host_request_polarity (*C enum*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_host_request_polarity.kLPI2C_HostRequestPinActiveHigh (*C enumerator*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_host_request_polarity.kLPI2C_HostRequestPinActiveLow (*C enumerator*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_host_request_source (*C enum*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_host_request_source.kLPI2C_HostRequestExternalPin (*C enumerator*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_host_request_source.kLPI2C_HostRequestInputTrigger (*C enumerator*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config (*C struct*), 625
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.baudRate_Hz (*C var*), 625
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.busIdleTimeout_ns (*C var*), 625
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.debugEnable (*C var*), 625
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.enableDoze (*C var*), 625
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.enableMaster (*C var*), 625
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.hostRequest (*C struct*), 627
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.hostRequest (*C var*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.hostRequest.enable (*C var*), 627
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.hostRequest.polarity (*C var*), 628
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.hostRequest.source (*C var*), 628
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.ignoreAck (*C var*), 625
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.pinConfig (*C var*), 625
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.pinLowTimeout_ns (*C var*), 625
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.sclGlitchFilterWidth_ns (*C var*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_master_config.sdaGlitchFilterWidth_ns (*C var*), 625
drivers_RT_RT700_MIMXRT798S._lpi2c_master_edma_handle (*C struct*), 629
drivers_RT_RT700_MIMXRT798S._lpi2c_master_edma_handle.base (*C var*), 630
drivers_RT_RT700_MIMXRT798S._lpi2c_master_edma_handle.commandBuffer (*C var*), 630
drivers_RT_RT700_MIMXRT798S._lpi2c_master_edma_handle.completionCallback (*C var*), 630
drivers_RT_RT700_MIMXRT798S._lpi2c_master_edma_handle.isBusy (*C var*), 630
drivers_RT_RT700_MIMXRT798S._lpi2c_master_edma_handle.nbytes (*C var*), 630
drivers_RT_RT700_MIMXRT798S._lpi2c_master_edma_handle.rx (*C var*), 630
drivers_RT_RT700_MIMXRT798S._lpi2c_master_edma_handle.tcds (*C var*), 630
drivers_RT_RT700_MIMXRT798S._lpi2c_master_edma_handle.transfer (*C var*), 630
drivers_RT_RT700_MIMXRT798S._lpi2c_master_edma_handle.tx (*C var*), 630
drivers_RT_RT700_MIMXRT798S._lpi2c_master_edma_handle.userData (*C var*), 630
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags (*C enum*), 620
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterArbitrationLostFlag (*C enumerator*), 621
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterBusBusyFlag (*C enumerator*), 621
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterBusyFlag (*C enumerator*), 621

drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterClearFlags (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterDataMatchFlag (*C enumerator*), 621
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterEndOfPacketFlag (*C enumerator*), 621
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterErrorFlags (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterFifoErrFlag (*C enumerator*), 621
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterIrqFlags (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterNackDetectFlag (*C enumerator*), 621
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterPinLowTimeoutFlag (*C enumerator*), 621
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterRxReadyFlag (*C enumerator*), 621
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterStopDetectFlag (*C enumerator*), 621
drivers_RT_RT700_MIMXRT798S._lpi2c_master_flags.kLPI2C_MasterTxReadyFlag (*C enumerator*), 621
drivers_RT_RT700_MIMXRT798S._lpi2c_master_handle (*C struct*), 627
drivers_RT_RT700_MIMXRT798S._lpi2c_master_handle.buf (*C var*), 627
drivers_RT_RT700_MIMXRT798S._lpi2c_master_handle.chunkSize (*C var*), 627
drivers_RT_RT700_MIMXRT798S._lpi2c_master_handle.commandBuffer (*C var*), 627
drivers_RT_RT700_MIMXRT798S._lpi2c_master_handle.completionCallback (*C var*), 627
drivers_RT_RT700_MIMXRT798S._lpi2c_master_handle.remainingBytes (*C var*), 627
drivers_RT_RT700_MIMXRT798S._lpi2c_master_handle.state (*C var*), 627
drivers_RT_RT700_MIMXRT798S._lpi2c_master_handle.transfer (*C var*), 627
drivers_RT_RT700_MIMXRT798S._lpi2c_master_handle.userData (*C var*), 627
drivers_RT_RT700_MIMXRT798S._lpi2c_master_pin_config (*C enum*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_master_pin_config.kLPI2C_2PinOpenDrain (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_master_pin_config.kLPI2C_2PinOpenDrainWithSeparateSlave (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_master_pin_config.kLPI2C_2PinOutputOnly (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_master_pin_config.kLPI2C_2PinOutputOnlyWithSeparateSlave (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_master_pin_config.kLPI2C_2PinPushPull (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_master_pin_config.kLPI2C_2PinPushPullWithSeparateSlave (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_master_pin_config.kLPI2C_4PinPushPull (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_master_pin_config.kLPI2C_4PinPushPullWithInvertedOutput (*C enumerator*), 622
drivers_RT_RT700_MIMXRT798S._lpi2c_master_transfer (*C struct*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_master_transfer_flags (*C enum*), 623
drivers_RT_RT700_MIMXRT798S._lpi2c_master_transfer_flags.kLPI2C_TransferDefaultFlag (*C enumerator*), 624
drivers_RT_RT700_MIMXRT798S._lpi2c_master_transfer_flags.kLPI2C_TransferNoStartFlag (*C enumerator*), 624
drivers_RT_RT700_MIMXRT798S._lpi2c_master_transfer_flags.kLPI2C_TransferNoStopFlag (*C enumerator*), 624
drivers_RT_RT700_MIMXRT798S._lpi2c_master_transfer.data (*C var*), 627
drivers_RT_RT700_MIMXRT798S._lpi2c_master_transfer.dataSize (*C var*), 627

drivers_RT_RT700_MIMXRT798S._lpi2c_master_transfer.direction (*C var*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_master_transfer.flags (*C var*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_master_transfer.slaveAddress (*C var*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_master_transfer.subaddress (*C var*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_master_transfer.subaddressSize (*C var*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_match_config (*C struct*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_match_config.match0 (*C var*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_match_config.match1 (*C var*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_match_config.matchMode (*C var*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_match_config.rxDataMatchOnly (*C var*), 626
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_address_match (*C enum*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_address_match.kLPI2C_MatchAddress0 (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_address_match.kLPI2C_MatchAddress0OrAddress1 (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_address_match.kLPI2C_MatchAddress0ThroughAddress1 (*C enumerator*), 638
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config (*C struct*), 639
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.address0 (*C var*), 639
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.address1 (*C var*), 639
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.addressMatchMode (*C var*), 639
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.clockHoldTime_ns (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.dataValidDelay_ns (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.enableGeneralCall (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.enableReceivedAddressRead (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.enableSlave (*C var*), 639
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.filterDozeEnable (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.filterEnable (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.ignoreAck (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.sclGlitchFilterWidth_ns (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.sclStall (*C struct*), 641
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.sclStall.enableAck (*C var*), 641
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.sclStall.enableAddress (*C var*), 642
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.sclStall.enableRx (*C var*), 642
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.sclStall.enableTx (*C var*), 641
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_config.sdaGlitchFilterWidth_ns (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags (*C enum*), 636
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveAddressMatch0Flag (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveAddressMatch1Flag (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveAddressValidFlag (*C enumerator*), 636
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveBitErrFlag (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveBusBusyFlag (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveBusyFlag (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveClearFlags (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveErrorFlags (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveFifoErrFlag (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveGeneralCallFlag (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveIrqFlags (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveRepeatedStartDetectFlag (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveRxReadyFlag (*C enumerator*), 636

drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveStopDetectFlag (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveTransmitAckFlag (*C enumerator*), 637
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_flags.kLPI2C_SlaveTxReadyFlag (*C enumerator*), 636
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_handle (*C struct*), 641
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_handle.callback (*C var*), 641
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_handle.eventMask (*C var*), 641
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_handle.isBusy (*C var*), 641
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_handle.transfer (*C var*), 641
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_handle.transferredCount (*C var*), 641
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_handle.userData (*C var*), 641
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_handle.wasTransmit (*C var*), 641
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer (*C struct*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer_event (*C enum*), 638
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer_event.kLPI2C_SlaveAddressMatchEvent (*C enumerator*), 638
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer_event.kLPI2C_SlaveAllEvents (*C enumerator*), 638
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer_event.kLPI2C_SlaveCompletionEvent (*C enumerator*), 638
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer_event.kLPI2C_SlaveReceiveEvent (*C enumerator*), 638
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer_event.kLPI2C_SlaveRepeatedStartEvent (*C enumerator*), 638
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer_event.kLPI2C_SlaveTransmitAckEvent (*C enumerator*), 638
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer_event.kLPI2C_SlaveTransmitEvent (*C enumerator*), 638
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer.completionStatus (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer.data (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer.dataSize (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer.event (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer.receivedAddress (*C var*), 640
drivers_RT_RT700_MIMXRT798S._lpi2c_slave_transfer.transferredCount (*C var*), 641
drivers_RT_RT700_MIMXRT798S._lpspi_clock_phase (*C enum*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_clock_phase.kLPSPI_ClockPhaseFirstEdge (*C enumerator*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_clock_phase.kLPSPI_ClockPhaseSecondEdge (*C enumerator*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_clock_polarity (*C enum*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_clock_polarity.kLPSPI_ClockPolarityActiveHigh (*C enumerator*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_clock_polarity.kLPSPI_ClockPolarityActiveLow (*C enumerator*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_data_out_config (*C enum*), 657
drivers_RT_RT700_MIMXRT798S._lpspi_data_out_config.kLpspiDataOutRetained (*C enumerator*), 657
drivers_RT_RT700_MIMXRT798S._lpspi_data_out_config.kLpspiDataOutTristate (*C enumerator*), 657
drivers_RT_RT700_MIMXRT798S._lpspi_delay_type (*C enum*), 657
drivers_RT_RT700_MIMXRT798S._lpspi_delay_type.kLPSPI_BetweenTransfer (*C enumerator*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_delay_type.kLPSPI_LastSckToPcs (*C enumerator*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_delay_type.kLPSPI_PcsToSck (*C enumerator*), 657
drivers_RT_RT700_MIMXRT798S._lpspi_dma_enable (*C enum*), 654
drivers_RT_RT700_MIMXRT798S._lpspi_dma_enable.kLPSPI_RxDmaEnable (*C enumerator*), 654

drivers_RT_RT700_MIMXRT798S._lpspi_dma_enable.kLPSPI_TxDmaEnable (*C enumerator*), 654
drivers_RT_RT700_MIMXRT798S._lpspi_flags (*C enum*), 652
drivers_RT_RT700_MIMXRT798S._lpspi_flags.kLPSPI_AllStatusFlag (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_flags.kLPSPI_DataMatchFlag (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_flags.kLPSPI_FrameCompleteFlag (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_flags.kLPSPI_ModuleBusyFlag (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_flags.kLPSPI_ReceiveErrorFlag (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_flags.kLPSPI_RxDataReadyFlag (*C enumerator*), 652
drivers_RT_RT700_MIMXRT798S._lpspi_flags.kLPSPI_TransferCompleteFlag (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_flags.kLPSPI_TransmitErrorFlag (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_flags.kLPSPI_TxDataRequestFlag (*C enumerator*), 652
drivers_RT_RT700_MIMXRT798S._lpspi_flags.kLPSPI_WordCompleteFlag (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_host_request_select (*C enum*), 656
drivers_RT_RT700_MIMXRT798S._lpspi_host_request_select.kLPSPI_HostReqExtPin (*C enumerator*), 656
drivers_RT_RT700_MIMXRT798S._lpspi_host_request_select.kLPSPI_HostReqInternalTrigger (*C enumerator*), 656
drivers_RT_RT700_MIMXRT798S._lpspi_interrupt_enable (*C enum*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_interrupt_enable.kLPSPI_AllInterruptEnable (*C enumerator*), 654
drivers_RT_RT700_MIMXRT798S._lpspi_interrupt_enable.kLPSPI_DataMatchInterruptEnable (*C enumerator*), 654
drivers_RT_RT700_MIMXRT798S._lpspi_interrupt_enable.kLPSPI_FrameCompleteInterruptEnable (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_interrupt_enable.kLPSPI_ReceiveErrorInterruptEnable (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_interrupt_enable.kLPSPI_RxInterruptEnable (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_interrupt_enable.kLPSPI_TransferCompleteInterruptEnable (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_interrupt_enable.kLPSPI_TransmitErrorInterruptEnable (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_interrupt_enable.kLPSPI_TxInterruptEnable (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_interrupt_enable.kLPSPI_WordCompleteInterruptEnable (*C enumerator*), 653
drivers_RT_RT700_MIMXRT798S._lpspi_master_config (*C struct*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.baudRate (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.betweenTransferDelayInNanoSec (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.bitsPerFrame (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.cpha (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.cpol (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.dataOutConfig (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.direction (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.enableInputDelay (*C var*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.lastSckToPcsDelayInNanoSec (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.pcsActiveHighOrLow (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.pcsFunc (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.pcsToSckDelayInNanoSec (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.pinCfg (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_config.whichPcs (*C var*), 662
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle (*C struct*), 670
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.callback (*C var*), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.edmaRxRegToRxDataHandle (*C var*), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.edmaTxDataToTxRegHandle (*C var*), 671

drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.isByteSwap (C var), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.isPcsContinuous (C var), 670
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.lastBytesRxWatermark (C var), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.lpspiSoftwareTCD (C var), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.nbytes (C var), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.oneFifoBlockRxWatermark (C var), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.rxBuffIfNull (C var), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.state (C var), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.totalByteCount (C var), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.transmitCommand (C var), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.txBuffIfNull (C var), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_edma_handle.userData (C var), 671
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle (C struct), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.bytesEachRead (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.bytesEachWrite (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.bytesPerFrame (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.callback (C var), 665
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.fifoSize (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.isByteSwap (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.isPcsContinuous (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.isTxMask (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.readRegRemainingTimes (C var), 665
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.rxData (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.rxRemainingByteCount (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.rxWatermark (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.state (C var), 665
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.totalByteCount (C var), 665
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.txBuffIfNull (C var), 665
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.txData (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.txRemainingByteCount (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.userData (C var), 665
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.writeRegRemainingTimes (C var), 665
drivers_RT_RT700_MIMXRT798S._lpspi_master_handle.writeTcrInIsr (C var), 664
drivers_RT_RT700_MIMXRT798S._lpspi_master_slave_mode (C enum), 654
drivers_RT_RT700_MIMXRT798S._lpspi_master_slave_mode.kLPSPI_Master (C enumerator), 654
drivers_RT_RT700_MIMXRT798S._lpspi_master_slave_mode.kLPSPI_Slave (C enumerator), 654
drivers_RT_RT700_MIMXRT798S._lpspi_match_config (C enum), 656
drivers_RT_RT700_MIMXRT798S._lpspi_match_config.kLPSI_1stWordAndM1EqualsM0andM1 (C enumerator), 656
drivers_RT_RT700_MIMXRT798S._lpspi_match_config.kLPSI_1stWordEqualsM0and2ndWordEqualsM1 (C enumerator), 656
drivers_RT_RT700_MIMXRT798S._lpspi_match_config.kLPSI_1stWordEqualsM0orM1 (C enumerator), 656
drivers_RT_RT700_MIMXRT798S._lpspi_match_config.kLPSI_AnyWordAndM1EqualsM0andM1 (C enumerator), 656
drivers_RT_RT700_MIMXRT798S._lpspi_match_config.kLPSI_AnyWordEqualsM0andNxtWordEqualsM1 (C enumerator), 656
drivers_RT_RT700_MIMXRT798S._lpspi_match_config.kLPSI_AnyWordEqualsM0orM1 (C enumerator), 656
drivers_RT_RT700_MIMXRT798S._lpspi_match_config.kLPSI_MatchDisabled (C enumerator), 656
drivers_RT_RT700_MIMXRT798S._lpspi_pcs_function_config (C enum), 657
drivers_RT_RT700_MIMXRT798S._lpspi_pcs_function_config.kLPSPI_PcsAsCs (C enumerator), 657
drivers_RT_RT700_MIMXRT798S._lpspi_pcs_function_config.kLPSPI_PcsAsData (C enumerator), 657

drivers_RT_RT700_MIMXRT798S._lpspi_pcs_polarity (*C enum*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_pcs_polarity_config (*C enum*), 654
drivers_RT_RT700_MIMXRT798S._lpspi_pcs_polarity_config.kLPSPI_PcsActiveHigh (*C enumerator*), 654
drivers_RT_RT700_MIMXRT798S._lpspi_pcs_polarity_config.kLPSPI_PcsActiveLow (*C enumerator*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_pcs_polarity.kLPSPI_Pcs0ActiveLow (*C enumerator*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_pcs_polarity.kLPSPI_Pcs1ActiveLow (*C enumerator*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_pcs_polarity.kLPSPI_Pcs2ActiveLow (*C enumerator*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_pcs_polarity.kLPSPI_Pcs3ActiveLow (*C enumerator*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_pcs_polarity.kLPSPI_PcsAllActiveLow (*C enumerator*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_pin_config (*C enum*), 656
drivers_RT_RT700_MIMXRT798S._lpspi_pin_config.kLPSPI_SdiInSdiOut (*C enumerator*), 657
drivers_RT_RT700_MIMXRT798S._lpspi_pin_config.kLPSPI_SdiInSdoOut (*C enumerator*), 656
drivers_RT_RT700_MIMXRT798S._lpspi_pin_config.kLPSPI_SdoInSdiOut (*C enumerator*), 657
drivers_RT_RT700_MIMXRT798S._lpspi_pin_config.kLPSPI_SdoInSdoOut (*C enumerator*), 657
drivers_RT_RT700_MIMXRT798S._lpspi_shift_direction (*C enum*), 655
drivers_RT_RT700_MIMXRT798S._lpspi_shift_direction.kLPSPI_LsbFirst (*C enumerator*), 656
drivers_RT_RT700_MIMXRT798S._lpspi_shift_direction.kLPSPI_MsbFirst (*C enumerator*), 656
drivers_RT_RT700_MIMXRT798S._lpspi_slave_config (*C struct*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_slave_config.bitsPerFrame (*C var*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_slave_config.cpha (*C var*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_slave_config.cpol (*C var*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_slave_config.dataOutConfig (*C var*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_slave_config.direction (*C var*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_slave_config.pcsActiveHighOrLow (*C var*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_slave_config.pinCfg (*C var*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_slave_config.whichPcs (*C var*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle (*C struct*), 671
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle.callback (*C var*), 672
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle.edmaRxRegToRxDataHandle (*C var*), 672
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle.edmaTxDataToTxRegHandle (*C var*), 672
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle.lastBytesRxWatermark (*C var*), 672
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle.lpspiSoftwareTCD (*C var*), 672
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle.nbytes (*C var*), 672
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle.oneFifoBlockRxWatermark (*C var*), 672
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle.rxBuffIfNull (*C var*), 672
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle.state (*C var*), 672
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle.totalByteCount (*C var*), 672
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle.txBuffIfNull (*C var*), 672
drivers_RT_RT700_MIMXRT798S._lpspi_slave_edma_handle.userData (*C var*), 672
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle (*C struct*), 665
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.bytesEachRead (*C var*), 665
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.bytesEachWrite (*C var*), 665
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.callback (*C var*), 666
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.errorCount (*C var*), 666
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.fifoSize (*C var*), 665
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.isByteSwap (*C var*), 665
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.readRegRemainingTimes (*C var*), 666
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.rxData (*C var*), 665

drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.rxRemainingByteCount (*C var*), 666
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.rxWatermark (*C var*), 665
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.state (*C var*), 666
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.totalByteCount (*C var*), 666
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.txData (*C var*), 665
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.txRemainingByteCount (*C var*), 666
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.userData (*C var*), 666
drivers_RT_RT700_MIMXRT798S._lpspi_slave_handle.writeRegRemainingTimes (*C var*), 666
drivers_RT_RT700_MIMXRT798S._lpspi_transfer (*C struct*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_master (*C enum*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_master.kLPSPI_MasterByteSwap (*C enumerator*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcs0 (*C enumerator*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcs1 (*C enumerator*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcs2 (*C enumerator*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcs3 (*C enumerator*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_master.kLPSPI_MasterPcsContinuous (*C enumerator*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_master.kLPSPI_MasterWidth1 (*C enumerator*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_master.kLPSPI_MasterWidth2 (*C enumerator*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_master.kLPSPI_MasterWidth4 (*C enumerator*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_slave (*C enum*), 658
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_slave.kLPSPI_SlaveByteSwap (*C enumerator*), 659
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_slave.kLPSPI_SlavePcs0 (*C enumerator*), 659
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_slave.kLPSPI_SlavePcs1 (*C enumerator*), 659
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_slave.kLPSPI_SlavePcs2 (*C enumerator*), 659
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_config_flag_for_slave.kLPSPI_SlavePcs3 (*C enumerator*), 659
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_state (*C enum*), 659
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_state.kLPSPI_Busy (*C enumerator*), 659
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_state.kLPSPI_Error (*C enumerator*), 659
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_state.kLPSPI_Idle (*C enumerator*), 659
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_width (*C enum*), 657
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_width.kLPSPI_FourBitXfer (*C enumerator*), 657
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_width.kLPSPI_SingleBitXfer (*C enumerator*), 657
drivers_RT_RT700_MIMXRT798S._lpspi_transfer_width.kLPSPI_TwoBitXfer (*C enumerator*), 657
drivers_RT_RT700_MIMXRT798S._lpspi_transfer.configFlags (*C var*), 664
drivers_RT_RT700_MIMXRT798S._lpspi_transfer.dataSize (*C var*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_transfer.rxData (*C var*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_transfer.txData (*C var*), 663
drivers_RT_RT700_MIMXRT798S._lpspi_which_pcs_config (*C enum*), 654
drivers_RT_RT700_MIMXRT798S._lpspi_which_pcs_config.kLPSPI_Pcs0 (*C enumerator*), 654
drivers_RT_RT700_MIMXRT798S._lpspi_which_pcs_config.kLPSPI_Pcs1 (*C enumerator*), 654
drivers_RT_RT700_MIMXRT798S._lpspi_which_pcs_config.kLPSPI_Pcs2 (*C enumerator*), 654
drivers_RT_RT700_MIMXRT798S._lpspi_which_pcs_config.kLPSPI_Pcs3 (*C enumerator*), 654

drivers_RT_RT700_MIMXRT798S._lpuart_config (*C struct*), 692
drivers_RT_RT700_MIMXRT798S._lpuart_config.baudRate_Bps (*C var*), 692
drivers_RT_RT700_MIMXRT798S._lpuart_config.dataBitsCount (*C var*), 692
drivers_RT_RT700_MIMXRT798S._lpuart_config.enableRx (*C var*), 693
drivers_RT_RT700_MIMXRT798S._lpuart_config.enableRxRTS (*C var*), 692
drivers_RT_RT700_MIMXRT798S._lpuart_config.enableSingleWire (*C var*), 693
drivers_RT_RT700_MIMXRT798S._lpuart_config.enableTx (*C var*), 693
drivers_RT_RT700_MIMXRT798S._lpuart_config.enableTxCTS (*C var*), 692
drivers_RT_RT700_MIMXRT798S._lpuart_config.isMsb (*C var*), 692
drivers_RT_RT700_MIMXRT798S._lpuart_config.parityMode (*C var*), 692
drivers_RT_RT700_MIMXRT798S._lpuart_config.rtsDelay (*C var*), 693
drivers_RT_RT700_MIMXRT798S._lpuart_config.rxFifoWatermark (*C var*), 692
drivers_RT_RT700_MIMXRT798S._lpuart_config.rxIdleConfig (*C var*), 693
drivers_RT_RT700_MIMXRT798S._lpuart_config.rxIdleType (*C var*), 693
drivers_RT_RT700_MIMXRT798S._lpuart_config.stopBitCount (*C var*), 692
drivers_RT_RT700_MIMXRT798S._lpuart_config.timeoutConfig (*C var*), 693
drivers_RT_RT700_MIMXRT798S._lpuart_config.txCtsConfig (*C var*), 693
drivers_RT_RT700_MIMXRT798S._lpuart_config.txCtsSource (*C var*), 692
drivers_RT_RT700_MIMXRT798S._lpuart_config.txFifoWatermark (*C var*), 692
drivers_RT_RT700_MIMXRT798S._lpuart_data_bits (*C enum*), 685
drivers_RT_RT700_MIMXRT798S._lpuart_data_bits.kLPUART_EightDataBits (*C enumerator*), 685
drivers_RT_RT700_MIMXRT798S._lpuart_data_bits.kLPUART_SevenDataBits (*C enumerator*), 685
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle (*C struct*), 698
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.callback (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.edmaTcd (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.rxData (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.rxDataSizeAll (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.rxEdmaHandle (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.rxState (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.rxWatermarkValue (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.txCbMode (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.txData (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.txDataSizeAll (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.txEdmaHandle (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.txState (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_edma_handle.userData (*C var*), 699
drivers_RT_RT700_MIMXRT798S._lpuart_flags (*C enum*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_AllClearFlags (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_AllFlags (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_CtsStateChangeFlag (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_DataMatch1Flag (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_DataMatch2Flag (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_DcdStateChangeFlag (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_DsrStateChangeFlag (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_FramingErrorFlag (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_IdleLineFlag (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_LinBreakFlag (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_NoiseErrorFlag (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_ParityErrorFlag (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_RinStateChangeFlag (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_RxActiveEdgeFlag (*C enumerator*), 689

drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_RxActiveFlag (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_RxCounter0TimeoutFlag (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_RxCounter1TimeoutFlag (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_RxDataRegFullFlag (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_RxFifoEmptyFlag (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_RxFifoUnderflowFlag (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_RxOverrunFlag (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_TransmissionCompleteFlag (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_TxCounter0TimeoutFlag (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_TxCounter1TimeoutFlag (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_TxDataRegEmptyFlag (*C enumerator*), 689
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_TxFifoEmptyFlag (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_flags.kLPUART_TxFifoOverflowFlag (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_handle (*C struct*), 693
drivers_RT_RT700_MIMXRT798S._lpuart_handle.___unnamed82___ (*C union*), 695
drivers_RT_RT700_MIMXRT798S._lpuart_handle.___unnamed82___.txData (*C var*), 695
drivers_RT_RT700_MIMXRT798S._lpuart_handle.___unnamed82___.txData16 (*C var*), 695
drivers_RT_RT700_MIMXRT798S._lpuart_handle.___unnamed84___ (*C union*), 695
drivers_RT_RT700_MIMXRT798S._lpuart_handle.___unnamed84___.rxData (*C var*), 695
drivers_RT_RT700_MIMXRT798S._lpuart_handle.___unnamed84___.rxData16 (*C var*), 695
drivers_RT_RT700_MIMXRT798S._lpuart_handle.___unnamed86___ (*C union*), 695
drivers_RT_RT700_MIMXRT798S._lpuart_handle.___unnamed86___.rxRingBuffer (*C var*), 695
drivers_RT_RT700_MIMXRT798S._lpuart_handle.___unnamed86___.rxRingBuffer16 (*C var*), 695
drivers_RT_RT700_MIMXRT798S._lpuart_handle.callback (*C var*), 694
drivers_RT_RT700_MIMXRT798S._lpuart_handle.is16bitData (*C var*), 694
drivers_RT_RT700_MIMXRT798S._lpuart_handle.isSevenDataBits (*C var*), 694
drivers_RT_RT700_MIMXRT798S._lpuart_handle.rxDataSize (*C var*), 693
drivers_RT_RT700_MIMXRT798S._lpuart_handle.rxDataSizeAll (*C var*), 694
drivers_RT_RT700_MIMXRT798S._lpuart_handle.rxRingBufferHead (*C var*), 694
drivers_RT_RT700_MIMXRT798S._lpuart_handle.rxRingBufferSize (*C var*), 694
drivers_RT_RT700_MIMXRT798S._lpuart_handle.rxRingBufferTail (*C var*), 694
drivers_RT_RT700_MIMXRT798S._lpuart_handle.rxState (*C var*), 694
drivers_RT_RT700_MIMXRT798S._lpuart_handle.txDataSize (*C var*), 693
drivers_RT_RT700_MIMXRT798S._lpuart_handle.txDataSizeAll (*C var*), 693
drivers_RT_RT700_MIMXRT798S._lpuart_handle.txState (*C var*), 694
drivers_RT_RT700_MIMXRT798S._lpuart_handle.userData (*C var*), 694
drivers_RT_RT700_MIMXRT798S._lpuart_idle_config (*C enum*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_idle_config.kLPUART_IdleCharacter1 (*C enumerator*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_idle_config.kLPUART_IdleCharacter2 (*C enumerator*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_idle_config.kLPUART_IdleCharacter4 (*C enumerator*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_idle_config.kLPUART_IdleCharacter8 (*C enumerator*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_idle_config.kLPUART_IdleCharacter16 (*C enumerator*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_idle_config.kLPUART_IdleCharacter32 (*C enumerator*), 686

drivers_RT_RT700_MIMXRT798S._lpuart_idle_config.kLPUART_IdleCharacter64 (*C enumerator*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_idle_config.kLPUART_IdleCharacter128 (*C enumerator*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_idle_type_select (*C enum*), 685
drivers_RT_RT700_MIMXRT798S._lpuart_idle_type_select.kLPUART_IdleTypeStartBit (*C enumerator*), 685
drivers_RT_RT700_MIMXRT798S._lpuart_idle_type_select.kLPUART_IdleTypeStopBit (*C enumerator*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable (*C enum*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_AllInterruptEnable (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_CtsStateChangeInterruptEnable (*C enumerator*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_DataMatch1InterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_DataMatch2InterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_DcdStateChangeInterruptEnable (*C enumerator*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_DsrStateChangeInterruptEnable (*C enumerator*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_FramingErrorInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_IdleLineInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_LinBreakInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_NoiseErrorInterruptEnable (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_ParityErrorInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_RinStateChangeInterruptEnable (*C enumerator*), 686
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_RxActiveEdgeInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_RxCounter0TimeoutInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_RxCounter1TimeoutInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_RxDataRegFullInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_RxFifoUnderflowInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_RxOverrunInterruptEnable (*C enumerator*), 688
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_TransmissionCompleteInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_TxCounter0TimeoutInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_TxCounter1TimeoutInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_TxDataRegEmptyInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_interrupt_enable.kLPUART_TxFifoOverflowInterruptEnable (*C enumerator*), 687
drivers_RT_RT700_MIMXRT798S._lpuart_parity_mode (*C enum*), 684
drivers_RT_RT700_MIMXRT798S._lpuart_parity_mode.kLPUART_ParityDisabled (*C enumera-*

tor), 684
drivers_RT_RT700_MIMXRT798S._lpuart_parity_mode.kLPUART_ParityEven (C enumerator), 684
drivers_RT_RT700_MIMXRT798S._lpuart_parity_mode.kLPUART_ParityOdd (C enumerator), 684
drivers_RT_RT700_MIMXRT798S._lpuart_stop_bit_count (C enum), 685
drivers_RT_RT700_MIMXRT798S._lpuart_stop_bit_count.kLPUART_OneStopBit (C enumerator), 685
drivers_RT_RT700_MIMXRT798S._lpuart_stop_bit_count.kLPUART_TwoStopBit (C enumerator), 685
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_condition (C enum), 689
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_condition.kLPUART_TimeoutAfterCharacters (C enumerator), 690
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_condition.kLPUART_TimeoutAfterIdle (C enumerator), 690
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_condition.kLPUART_TimeoutAfterIdleBeforeExtended (C enumerator), 690
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_condition.kLPUART_TimeoutAfterNext (C enumerator), 690
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_config (C struct), 691
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_config.rxCounter0 (C var), 692
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_config.rxCounter1 (C var), 692
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_config.rxExtendedTimeoutValue (C var), 691
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_config.txCounter0 (C var), 692
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_config.txCounter1 (C var), 692
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_config.txExtendedTimeoutValue (C var), 691
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_counter_config (C struct), 691
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_counter_config.enableCounter (C var), 691
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_counter_config.timeoutCondition (C var), 691
drivers_RT_RT700_MIMXRT798S._lpuart_timeout_counter_config.timeoutValue (C var), 691
drivers_RT_RT700_MIMXRT798S._lpuart_transfer (C struct), 693
drivers_RT_RT700_MIMXRT798S._lpuart_transfer.___unnamed80___ (C union), 694
drivers_RT_RT700_MIMXRT798S._lpuart_transfer.___unnamed80___.data (C var), 694
drivers_RT_RT700_MIMXRT798S._lpuart_transfer.___unnamed80___.rxData (C var), 694
drivers_RT_RT700_MIMXRT798S._lpuart_transfer.___unnamed80___.rxData16 (C var), 694
drivers_RT_RT700_MIMXRT798S._lpuart_transfer.___unnamed80___.txData (C var), 694
drivers_RT_RT700_MIMXRT798S._lpuart_transfer.___unnamed80___.txData16 (C var), 694
drivers_RT_RT700_MIMXRT798S._lpuart_transfer.dataSize (C var), 693
drivers_RT_RT700_MIMXRT798S._lpuart_transmit_cts_config (C enum), 685
drivers_RT_RT700_MIMXRT798S._lpuart_transmit_cts_config.kLPUART_CtsSampleAtIdle (C enumerator), 685
drivers_RT_RT700_MIMXRT798S._lpuart_transmit_cts_config.kLPUART_CtsSampleAtStart (C enumerator), 685
drivers_RT_RT700_MIMXRT798S._lpuart_transmit_cts_source (C enum), 685
drivers_RT_RT700_MIMXRT798S._lpuart_transmit_cts_source.kLPUART_CtsSourceMatchResult (C enumerator), 685
drivers_RT_RT700_MIMXRT798S._lpuart_transmit_cts_source.kLPUART_CtsSourcePin (C enumerator), 685
drivers_RT_RT700_MIMXRT798S._lpuart_tx_callback_mode (C enum), 698
drivers_RT_RT700_MIMXRT798S._lpuart_tx_callback_mode.kLPUART_TxEdmaComplete (C enumerator), 698
drivers_RT_RT700_MIMXRT798S._lpuart_tx_callback_mode.kLPUART_TxFifoEmpty (C enumerator), 698
drivers_RT_RT700_MIMXRT798S._main_pll_mult (C enum), 243
drivers_RT_RT700_MIMXRT798S._main_pll_mult.kCLOCK_MainPllMult16 (C enumerator), 243
drivers_RT_RT700_MIMXRT798S._main_pll_mult.kCLOCK_MainPllMult17 (C enumerator), 243
drivers_RT_RT700_MIMXRT798S._main_pll_mult.kCLOCK_MainPllMult18 (C enumerator), 243
drivers_RT_RT700_MIMXRT798S._main_pll_mult.kCLOCK_MainPllMult19 (C enumerator), 243

drivers_RT_RT700_MIMXRT798S._main_pll_mult.kCLOCK_MainPllMult20 (*C enumerator*), 243
drivers_RT_RT700_MIMXRT798S._main_pll_mult.kCLOCK_MainPllMult21 (*C enumerator*), 243
drivers_RT_RT700_MIMXRT798S._main_pll_mult.kCLOCK_MainPllMult22 (*C enumerator*), 244
drivers_RT_RT700_MIMXRT798S._main_pll_src (*C enum*), 243
drivers_RT_RT700_MIMXRT798S._main_pll_src.kCLOCK_MainPllFro1Div8 (*C enumerator*), 243
drivers_RT_RT700_MIMXRT798S._main_pll_src.kCLOCK_MainPllOscClk (*C enumerator*), 243
drivers_RT_RT700_MIMXRT798S._mmu_ahb_access_type (*C enum*), 730
drivers_RT_RT700_MIMXRT798S._mmu_ahb_access_type.kMMU_ReadAccess (*C enumerator*), 730
drivers_RT_RT700_MIMXRT798S._mmu_ahb_access_type.kMMU_WriteAccess (*C enumerator*), 730
drivers_RT_RT700_MIMXRT798S._mmu_ahb_error_record (*C struct*), 733
drivers_RT_RT700_MIMXRT798S._mmu_ahb_error_record.accessType (*C var*), 733
drivers_RT_RT700_MIMXRT798S._mmu_ahb_error_record.errorType (*C var*), 733
drivers_RT_RT700_MIMXRT798S._mmu_ahb_error_record.failAddr (*C var*), 733
drivers_RT_RT700_MIMXRT798S._mmu_ahb_error_record.masterId (*C var*), 733
drivers_RT_RT700_MIMXRT798S._mmu_ahb_error_record.protectionInfo (*C var*), 733
drivers_RT_RT700_MIMXRT798S._mmu_ahb_error_type (*C enum*), 730
drivers_RT_RT700_MIMXRT798S._mmu_ahb_error_type.kMMU_AddrInvalidError (*C enumerator*), 731
drivers_RT_RT700_MIMXRT798S._mmu_ahb_error_type.kMMU_LUTInvalidError (*C enumerator*), 731
drivers_RT_RT700_MIMXRT798S._mmu_ahb_error_type.kMMU_ParityError (*C enumerator*), 731
drivers_RT_RT700_MIMXRT798S._mmu_config (*C struct*), 733
drivers_RT_RT700_MIMXRT798S._mmu_config.baseAddr (*C var*), 734
drivers_RT_RT700_MIMXRT798S._mmu_config.countOfAddr (*C var*), 734
drivers_RT_RT700_MIMXRT798S._mmu_config.enableMMP (*C var*), 734
drivers_RT_RT700_MIMXRT798S._mmu_config.invalidAddrMode (*C var*), 734
drivers_RT_RT700_MIMXRT798S._mmu_config.maxAddr (*C var*), 734
drivers_RT_RT700_MIMXRT798S._mmu_config.pageOpt (*C var*), 734
drivers_RT_RT700_MIMXRT798S._mmu_config.ptrPhyAddrArray (*C var*), 734
drivers_RT_RT700_MIMXRT798S._mmu_config.ptrVirtualAddrArray (*C var*), 734
drivers_RT_RT700_MIMXRT798S._mmu_config.unitSize (*C var*), 734
drivers_RT_RT700_MIMXRT798S._mmu_interrupt_enable (*C enum*), 729
drivers_RT_RT700_MIMXRT798S._mmu_interrupt_enable.kMMU_AddressInvalidInterruptEnable (*C enumerator*), 730
drivers_RT_RT700_MIMXRT798S._mmu_interrupt_enable.kMMU_AllInterruptsEnable (*C enumerator*), 730
drivers_RT_RT700_MIMXRT798S._mmu_interrupt_enable.kMMU_IpsReadErrorInterruptEnable (*C enumerator*), 730
drivers_RT_RT700_MIMXRT798S._mmu_interrupt_enable.kMMU_LockErrorInterruptEnable (*C enumerator*), 730
drivers_RT_RT700_MIMXRT798S._mmu_interrupt_enable.kMMU_LUTInvalidInterruptEnable (*C enumerator*), 729
drivers_RT_RT700_MIMXRT798S._mmu_interrupt_enable.kMMU_ParityErrorInterruptEnable (*C enumerator*), 730
drivers_RT_RT700_MIMXRT798S._mmu_invalid_address_mode (*C enum*), 732
drivers_RT_RT700_MIMXRT798S._mmu_invalid_address_mode.kMMU_InvalidAddrAllowed (*C enumerator*), 732
drivers_RT_RT700_MIMXRT798S._mmu_invalid_address_mode.kMMU_InvalidAddrRespondError (*C enumerator*), 732
drivers_RT_RT700_MIMXRT798S._mmu_lut_word (*C struct*), 733
drivers_RT_RT700_MIMXRT798S._mmu_lut_word.mappedValue (*C var*), 733
drivers_RT_RT700_MIMXRT798S._mmu_lut_word.valid (*C var*), 733
drivers_RT_RT700_MIMXRT798S._mmu_lut_write_operation (*C enum*), 731
drivers_RT_RT700_MIMXRT798S._mmu_lut_write_operation.kMMU_LUTBurstWrite (*C enumerator*), 731

drivers_RT_RT700_MIMXRT798S._mmu_lut_write_operation.kMMU_LUTSingleWrite (*C enumerator*), 731
drivers_RT_RT700_MIMXRT798S._mmu_page_option (*C enum*), 731
drivers_RT_RT700_MIMXRT798S._mmu_page_option.kMMU_PageOpt_1Page (*C enumerator*), 731
drivers_RT_RT700_MIMXRT798S._mmu_page_option.kMMU_PageOpt_4Page_High2KWordSelected (*C enumerator*), 731
drivers_RT_RT700_MIMXRT798S._mmu_page_option.kMMU_PageOpt_4Page_Low2KWordSelected (*C enumerator*), 731
drivers_RT_RT700_MIMXRT798S._mmu_page_option.kMMU_PageOpt_4Page_MidHigh2KWordSelected (*C enumerator*), 731
drivers_RT_RT700_MIMXRT798S._mmu_page_option.kMMU_PageOpt_4Page_MidLow2KWordSelected (*C enumerator*), 731
drivers_RT_RT700_MIMXRT798S._mmu_status_flags (*C enum*), 729
drivers_RT_RT700_MIMXRT798S._mmu_status_flags.kMMU_AddressInvalidFlag (*C enumerator*), 729
drivers_RT_RT700_MIMXRT798S._mmu_status_flags.kMMU_AllFlags (*C enumerator*), 729
drivers_RT_RT700_MIMXRT798S._mmu_status_flags.kMMU_BusyFlag (*C enumerator*), 729
drivers_RT_RT700_MIMXRT798S._mmu_status_flags.kMMU_IpsReadErrorFlag (*C enumerator*), 729
drivers_RT_RT700_MIMXRT798S._mmu_status_flags.kMMU_LockErrorFlag (*C enumerator*), 729
drivers_RT_RT700_MIMXRT798S._mmu_status_flags.kMMU_LUTInvalidFlag (*C enumerator*), 729
drivers_RT_RT700_MIMXRT798S._mmu_status_flags.kMMU_ParityErrorFlag (*C enumerator*), 729
drivers_RT_RT700_MIMXRT798S._mmu_unit_size (*C enum*), 730
drivers_RT_RT700_MIMXRT798S._mmu_unit_size.kMMU_UnitSize4KB (*C enumerator*), 730
drivers_RT_RT700_MIMXRT798S._mmu_unit_size.kMMU_UnitSize8KB (*C enumerator*), 730
drivers_RT_RT700_MIMXRT798S._mmu_unit_size.kMMU_UnitSize16KB (*C enumerator*), 730
drivers_RT_RT700_MIMXRT798S._mmu_unit_size.kMMU_UnitSize32KB (*C enumerator*), 730
drivers_RT_RT700_MIMXRT798S._mmu_unit_size.kMMU_UnitSize64KB (*C enumerator*), 730
drivers_RT_RT700_MIMXRT798S._mrt_chnl (*C enum*), 737
drivers_RT_RT700_MIMXRT798S._mrt_chnl.kMRT_Channel_0 (*C enumerator*), 737
drivers_RT_RT700_MIMXRT798S._mrt_chnl.kMRT_Channel_1 (*C enumerator*), 737
drivers_RT_RT700_MIMXRT798S._mrt_chnl.kMRT_Channel_2 (*C enumerator*), 737
drivers_RT_RT700_MIMXRT798S._mrt_chnl.kMRT_Channel_3 (*C enumerator*), 737
drivers_RT_RT700_MIMXRT798S._mrt_config (*C struct*), 738
drivers_RT_RT700_MIMXRT798S._mrt_config.enableMultiTask (*C var*), 739
drivers_RT_RT700_MIMXRT798S._mrt_interrupt_enable (*C enum*), 738
drivers_RT_RT700_MIMXRT798S._mrt_interrupt_enable.kMRT_TimerInterruptEnable (*C enumerator*), 738
drivers_RT_RT700_MIMXRT798S._mrt_status_flags (*C enum*), 738
drivers_RT_RT700_MIMXRT798S._mrt_status_flags.kMRT_TimerInterruptFlag (*C enumerator*), 738
drivers_RT_RT700_MIMXRT798S._mrt_status_flags.kMRT_TimerRunFlag (*C enumerator*), 738
drivers_RT_RT700_MIMXRT798S._mrt_timer_mode (*C enum*), 737
drivers_RT_RT700_MIMXRT798S._mrt_timer_mode.kMRT_OneShotMode (*C enumerator*), 738
drivers_RT_RT700_MIMXRT798S._mrt_timer_mode.kMRT_OneShotStallMode (*C enumerator*), 738
drivers_RT_RT700_MIMXRT798S._mrt_timer_mode.kMRT_RepeatMode (*C enumerator*), 738
drivers_RT_RT700_MIMXRT798S._mu_general_purpose_interrupt (*C enum*), 751
drivers_RT_RT700_MIMXRT798S._mu_general_purpose_interrupt.kMU_GeneralPurposeInterrupt0 (*C enumerator*), 751
drivers_RT_RT700_MIMXRT798S._mu_general_purpose_interrupt.kMU_GeneralPurposeInterrupt1 (*C enumerator*), 751
drivers_RT_RT700_MIMXRT798S._mu_general_purpose_interrupt.kMU_GeneralPurposeInterrupt2 (*C enumerator*), 751
drivers_RT_RT700_MIMXRT798S._mu_general_purpose_interrupt.kMU_GeneralPurposeInterrupt3

(*C enumerator*), 751
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable (*C enum*), 749
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_GenInt0InterruptEnable (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_GenInt1InterruptEnable (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_GenInt2InterruptEnable (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_GenInt3InterruptEnable (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_MuResetInterruptEnable (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_Rx0FullInterruptEnable (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_Rx1FullInterruptEnable (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_Rx2FullInterruptEnable (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_Rx3FullInterruptEnable (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_Tx0EmptyInterruptEnable (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_Tx1EmptyInterruptEnable (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_Tx2EmptyInterruptEnable (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_enable.kMU_Tx3EmptyInterruptEnable (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_trigger (*C enum*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_trigger.kMU_GenInt0InterruptTrigger (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_trigger.kMU_GenInt1InterruptTrigger (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_trigger.kMU_GenInt2InterruptTrigger (*C enumerator*), 750
drivers_RT_RT700_MIMXRT798S._mu_interrupt_trigger.kMU_GenInt3InterruptTrigger (*C enumerator*), 751
drivers_RT_RT700_MIMXRT798S._mu_msg_reg_index (*C enum*), 751
drivers_RT_RT700_MIMXRT798S._mu_msg_reg_index.kMU_MsgReg0 (*C enumerator*), 751
drivers_RT_RT700_MIMXRT798S._mu_msg_reg_index.kMU_MsgReg1 (*C enumerator*), 751
drivers_RT_RT700_MIMXRT798S._mu_msg_reg_index.kMU_MsgReg2 (*C enumerator*), 751
drivers_RT_RT700_MIMXRT798S._mu_msg_reg_index.kMU_MsgReg3 (*C enumerator*), 751
drivers_RT_RT700_MIMXRT798S._mu_status_flags (*C enum*), 748
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_EventPendingFlag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_FlagsUpdatingFlag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_GenInt0Flag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_GenInt1Flag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_GenInt2Flag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_GenInt3Flag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_GenIntPendingFlag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_MuInResetFlag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_MuResetInterruptFlag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_Rx0FullFlag (*C enumerator*), 748
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_Rx1FullFlag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_Rx2FullFlag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_Rx3FullFlag (*C enumerator*), 749

drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_RxFullPendingFlag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_Tx0EmptyFlag (*C enumerator*), 748
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_Tx1EmptyFlag (*C enumerator*), 748
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_Tx2EmptyFlag (*C enumerator*), 748
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_Tx3EmptyFlag (*C enumerator*), 748
drivers_RT_RT700_MIMXRT798S._mu_status_flags.kMU_TxEmptyPendingFlag (*C enumerator*), 749
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim (*C enum*), 244
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf0 (*C enumerator*), 244
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf2 (*C enumerator*), 244
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf4 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf6 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf8 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf10 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf12 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf14 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf16 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf18 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf20 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf22 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf24 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf26 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf28 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._osc32k_cap_trim.kCLOCK_Osc32kCapPf30 (*C enumerator*), 245
drivers_RT_RT700_MIMXRT798S._ostimer_flags (*C enum*), 758
drivers_RT_RT700_MIMXRT798S._ostimer_flags.kOSTIMER_MatchInterruptFlag (*C enumerator*), 758
drivers_RT_RT700_MIMXRT798S._pdm_channel_config (*C struct*), 769
drivers_RT_RT700_MIMXRT798S._pdm_channel_config.gain (*C var*), 770
drivers_RT_RT700_MIMXRT798S._pdm_channel_config.outputCutOffFreq (*C var*), 770
drivers_RT_RT700_MIMXRT798S._pdm_channel_enable_mask (*C enum*), 765
drivers_RT_RT700_MIMXRT798S._pdm_channel_enable_mask.kPDM_EnableChannel0 (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_channel_enable_mask.kPDM_EnableChannel1 (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_channel_enable_mask.kPDM_EnableChannel2 (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_channel_enable_mask.kPDM_EnableChannel3 (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_channel_enable_mask.kPDM_EnableChannelAll (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_config (*C struct*), 770

drivers_RT_RT700_MIMXRT798S._pdm_config.cicOverSampleRate (*C var*), 770
drivers_RT_RT700_MIMXRT798S._pdm_config.enableDoze (*C var*), 770
drivers_RT_RT700_MIMXRT798S._pdm_config.enableFilterBypass (*C var*), 770
drivers_RT_RT700_MIMXRT798S._pdm_config.fifoWatermark (*C var*), 770
drivers_RT_RT700_MIMXRT798S._pdm_config.qualityMode (*C var*), 770
drivers_RT_RT700_MIMXRT798S._pdm_data_width (*C enum*), 769
drivers_RT_RT700_MIMXRT798S._pdm_data_width.kPDM_DataWwidth24 (*C enumerator*), 769
drivers_RT_RT700_MIMXRT798S._pdm_data_width.kPDM_DataWwidth32 (*C enumerator*), 769
drivers_RT_RT700_MIMXRT798S._pdm_dc_remover (*C enum*), 766
drivers_RT_RT700_MIMXRT798S._pdm_dc_remover.kPDM_DcRemoverBypass (*C enumerator*), 767
drivers_RT_RT700_MIMXRT798S._pdm_dc_remover.kPDM_DcRemoverCutOff13Hz (*C enumerator*), 767
drivers_RT_RT700_MIMXRT798S._pdm_dc_remover.kPDM_DcRemoverCutOff20Hz (*C enumerator*), 767
drivers_RT_RT700_MIMXRT798S._pdm_dc_remover.kPDM_DcRemoverCutOff40Hz (*C enumerator*), 767
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain (*C enum*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain0 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain1 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain2 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain3 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain4 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain5 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain6 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain7 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain8 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain9 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain10 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain11 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain12 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain13 (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain14 (*C enumerator*), 769
drivers_RT_RT700_MIMXRT798S._pdm_df_output_gain.kPDM_DfOutputGain15 (*C enumerator*), 769
drivers_RT_RT700_MIMXRT798S._pdm_df_quality_mode (*C enum*), 767
drivers_RT_RT700_MIMXRT798S._pdm_df_quality_mode.kPDM_QualityModeHigh (*C enumerator*), 767
drivers_RT_RT700_MIMXRT798S._pdm_df_quality_mode.kPDM_QualityModeLow (*C enumerator*), 767
drivers_RT_RT700_MIMXRT798S._pdm_df_quality_mode.kPDM_QualityModeMedium (*C enumerator*), 767
drivers_RT_RT700_MIMXRT798S._pdm_df_quality_mode.kPDM_QualityModeVeryLow0 (*C enu-*

merator), 767
drivers_RT_RT700_MIMXRT798S._pdm_df_quality_mode.kPDM_QualityModeVeryLow1 (*C enumerator*), 767
drivers_RT_RT700_MIMXRT798S._pdm_df_quality_mode.kPDM_QualityModeVeryLow2 (*C enumerator*), 767
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle (*C struct*), 774
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.callback (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.channelNums (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.count (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.dmaHandle (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.endChannel (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.interleaveType (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.isLoopTransfer (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.receivedBytes (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.state (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.tcd (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.tcdDriver (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.tcdNum (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.tcdUsedNum (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.tcdUser (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_handle.userData (*C var*), 775
drivers_RT_RT700_MIMXRT798S._pdm_edma_multi_channel_interleave (*C enum*), 774
drivers_RT_RT700_MIMXRT798S._pdm_edma_multi_channel_interleave.kPDM_EDMAMultiChannelInterleavePer (*C enumerator*), 774
drivers_RT_RT700_MIMXRT798S._pdm_edma_multi_channel_interleave.kPDM_EDMAMultiChannelInterleavePer (*C enumerator*), 774
drivers_RT_RT700_MIMXRT798S._pdm_edma_transfer (*C struct*), 774
drivers_RT_RT700_MIMXRT798S._pdm_edma_transfer.data (*C var*), 774
drivers_RT_RT700_MIMXRT798S._pdm_edma_transfer.dataSize (*C var*), 774
drivers_RT_RT700_MIMXRT798S._pdm_edma_transfer.linkTransfer (*C var*), 774
drivers_RT_RT700_MIMXRT798S._pdm_fifo_status (*C enum*), 765
drivers_RT_RT700_MIMXRT798S._pdm_fifo_status.kPDM_FifoStatusOverflowCh0 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_fifo_status.kPDM_FifoStatusOverflowCh1 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_fifo_status.kPDM_FifoStatusOverflowCh2 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_fifo_status.kPDM_FifoStatusOverflowCh3 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_fifo_status.kPDM_FifoStatusUnderflowCh0 (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_fifo_status.kPDM_FifoStatusUnderflowCh1 (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_fifo_status.kPDM_FifoStatusUnderflowCh2 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_fifo_status.kPDM_FifoStatusUnderflowCh3 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_handle (*C struct*), 770
drivers_RT_RT700_MIMXRT798S._pdm_handle.callback (*C var*), 771
drivers_RT_RT700_MIMXRT798S._pdm_handle.channelNums (*C var*), 771
drivers_RT_RT700_MIMXRT798S._pdm_handle.format (*C var*), 771
drivers_RT_RT700_MIMXRT798S._pdm_handle.pdmQueue (*C var*), 771
drivers_RT_RT700_MIMXRT798S._pdm_handle.queueDriver (*C var*), 771
drivers_RT_RT700_MIMXRT798S._pdm_handle.queueUser (*C var*), 771
drivers_RT_RT700_MIMXRT798S._pdm_handle.startChannel (*C var*), 771
drivers_RT_RT700_MIMXRT798S._pdm_handle.state (*C var*), 770
drivers_RT_RT700_MIMXRT798S._pdm_handle.transferSize (*C var*), 771
drivers_RT_RT700_MIMXRT798S._pdm_handle.userData (*C var*), 771

drivers_RT_RT700_MIMXRT798S._pdm_handle.watermark (*C var*), 771
drivers_RT_RT700_MIMXRT798S._pdm_internal_status (*C enum*), 764
drivers_RT_RT700_MIMXRT798S._pdm_internal_status.kPDM_StatusCh0FifoDataAvaliable (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_internal_status.kPDM_StatusCh1FifoDataAvaliable (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_internal_status.kPDM_StatusCh2FifoDataAvaliable (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_internal_status.kPDM_StatusCh3FifoDataAvaliable (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_internal_status.kPDM_StatusDfBusyFlag (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_internal_status.kPDM_StatusFrequencyLow (*C enumerator*), 765
drivers_RT_RT700_MIMXRT798S._pdm_interrupt_enable (*C enum*), 764
drivers_RT_RT700_MIMXRT798S._pdm_interrupt_enable.kPDM_ErrorInterruptEnable (*C enumerator*), 764
drivers_RT_RT700_MIMXRT798S._pdm_interrupt_enable.kPDM_FIFOInterruptEnable (*C enumerator*), 764
drivers_RT_RT700_MIMXRT798S._pdm_qulaity_mode_k_factor (*C enum*), 767
drivers_RT_RT700_MIMXRT798S._pdm_qulaity_mode_k_factor.kPDM_QualityModeHighKFactor (*C enumerator*), 767
drivers_RT_RT700_MIMXRT798S._pdm_qulaity_mode_k_factor.kPDM_QualityModeLowKFactor (*C enumerator*), 767
drivers_RT_RT700_MIMXRT798S._pdm_qulaity_mode_k_factor.kPDM_QualityModeMediumKFactor (*C enumerator*), 767
drivers_RT_RT700_MIMXRT798S._pdm_qulaity_mode_k_factor.kPDM_QualityModeVeryLow2KFactor (*C enumerator*), 768
drivers_RT_RT700_MIMXRT798S._pdm_range_status (*C enum*), 766
drivers_RT_RT700_MIMXRT798S._pdm_range_status.kPDM_RangeStatusOverFlowCh0 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_range_status.kPDM_RangeStatusOverFlowCh1 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_range_status.kPDM_RangeStatusOverFlowCh2 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_range_status.kPDM_RangeStatusOverFlowCh3 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_range_status.kPDM_RangeStatusUnderFlowCh0 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_range_status.kPDM_RangeStatusUnderFlowCh1 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_range_status.kPDM_RangeStatusUnderFlowCh2 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_range_status.kPDM_RangeStatusUnderFlowCh3 (*C enumerator*), 766
drivers_RT_RT700_MIMXRT798S._pdm_transfer (*C struct*), 770
drivers_RT_RT700_MIMXRT798S._pdm_transfer.data (*C var*), 770
drivers_RT_RT700_MIMXRT798S._pdm_transfer.dataSize (*C var*), 770
drivers_RT_RT700_MIMXRT798S._pint_int (*C enum*), 776
drivers_RT_RT700_MIMXRT798S._pint_int.kPINT_PinInt0 (*C enumerator*), 776
drivers_RT_RT700_MIMXRT798S._pint_pin_enable (*C enum*), 776
drivers_RT_RT700_MIMXRT798S._pint_pin_enable.kPINT_PinIntEnableBothEdges (*C enumerator*), 776
drivers_RT_RT700_MIMXRT798S._pint_pin_enable.kPINT_PinIntEnableFallEdge (*C enumerator*), 776
drivers_RT_RT700_MIMXRT798S._pint_pin_enable.kPINT_PinIntEnableHighLevel (*C enumerator*), 776
drivers_RT_RT700_MIMXRT798S._pint_pin_enable.kPINT_PinIntEnableLowLevel (*C enumera-*

tor), 776
drivers_RT_RT700_MIMXRT798S._pint_pin_enable.kPINT_PinIntEnableNone (*C enumerator*), 776
drivers_RT_RT700_MIMXRT798S._pint_pin_enable.kPINT_PinIntEnableRiseEdge (*C enumerator*), 776
drivers_RT_RT700_MIMXRT798S._pint_pmatch_bslice (*C enum*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_bslice_cfg (*C enum*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_bslice_cfg.kPINT_PatternMatchAlways (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_bslice_cfg.kPINT_PatternMatchBothEdges (*C enumerator*), 778
drivers_RT_RT700_MIMXRT798S._pint_pmatch_bslice_cfg.kPINT_PatternMatchHigh (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_bslice_cfg.kPINT_PatternMatchLow (*C enumerator*), 778
drivers_RT_RT700_MIMXRT798S._pint_pmatch_bslice_cfg.kPINT_PatternMatchNever (*C enumerator*), 778
drivers_RT_RT700_MIMXRT798S._pint_pmatch_bslice_cfg.kPINT_PatternMatchStickyBothEdges (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_bslice_cfg.kPINT_PatternMatchStickyFall (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_bslice_cfg.kPINT_PatternMatchStickyRise (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_bslice.kPINT_PatternMatchBSlice0 (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_cfg (*C struct*), 785
drivers_RT_RT700_MIMXRT798S._pint_pmatch_input_src (*C enum*), 776
drivers_RT_RT700_MIMXRT798S._pint_pmatch_input_src.kPINT_PatternMatchInp0Src (*C enumerator*), 776
drivers_RT_RT700_MIMXRT798S._pint_pmatch_input_src.kPINT_PatternMatchInp1Src (*C enumerator*), 776
drivers_RT_RT700_MIMXRT798S._pint_pmatch_input_src.kPINT_PatternMatchInp2Src (*C enumerator*), 776
drivers_RT_RT700_MIMXRT798S._pint_pmatch_input_src.kPINT_PatternMatchInp3Src (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_input_src.kPINT_PatternMatchInp4Src (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_input_src.kPINT_PatternMatchInp5Src (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_input_src.kPINT_PatternMatchInp6Src (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_input_src.kPINT_PatternMatchInp7Src (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_input_src.kPINT_SecPatternMatchInp0Src (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_pmatch_input_src.kPINT_SecPatternMatchInp1Src (*C enumerator*), 777
drivers_RT_RT700_MIMXRT798S._pint_status (*C struct*), 785
drivers_RT_RT700_MIMXRT798S._pmic_mode_reg (*C enum*), 803
drivers_RT_RT700_MIMXRT798S._pmic_mode_reg.kCfg_Run (*C enumerator*), 803
drivers_RT_RT700_MIMXRT798S._pmic_mode_reg.kCfg_Sleep (*C enumerator*), 803
drivers_RT_RT700_MIMXRT798S._pngdec_color_type (*C enum*), 787
drivers_RT_RT700_MIMXRT798S._pngdec_color_type.kPNGDEC_PixelFormatLUT8 (*C enumerator*), 787
drivers_RT_RT700_MIMXRT798S._pngdec_color_type.kPNGDEC_PixelFormatRGB16_16_16 (*C enumerator*), 787
drivers_RT_RT700_MIMXRT798S._pngdec_color_type.kPNGDEC_PixelFormatRGB888 (*C enumerator*), 787

drivers_RT_RT700_MIMXRT798S._pngdec_color_type.kPNGDEC_PixelFormatRGBA16_16_16_16 (*C enumerator*), 787
drivers_RT_RT700_MIMXRT798S._pngdec_color_type.kPNGDEC_PixelFormatRGBA8888 (*C enumerator*), 787
drivers_RT_RT700_MIMXRT798S._pngdec_color_type.kPNGDEC_PixelFormatY8 (*C enumerator*), 787
drivers_RT_RT700_MIMXRT798S._pngdec_color_type.kPNGDEC_PixelFormatYA88 (*C enumerator*), 787
drivers_RT_RT700_MIMXRT798S._pngdec_config (*C struct*), 791
drivers_RT_RT700_MIMXRT798S._pngdec_endian_mode (*C enum*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_endian_mode.kPNGDEC_BigEndian (*C enumerator*), 787
drivers_RT_RT700_MIMXRT798S._pngdec_endian_mode.kPNGDEC_LittleEndian (*C enumerator*), 787
drivers_RT_RT700_MIMXRT798S._pngdec_image (*C struct*), 791
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable (*C enum*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable.kPNGDEC_BitDepthErrorFlag (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable.kPNGDEC_BtypeErrorFlag (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable.kPNGDEC_ChecksumErrorFlag (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable.kPNGDEC_CrcErrorFlag (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable.kPNGDEC_DecodeAncillaryDoneFlag (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable.kPNGDEC_DecodePixelDoneFlag (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable.kPNGDEC_EncodeDataDoneFlag (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable.kPNGDEC_ErrorFlags (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable.kPNGDEC_HeaderErrorFlag (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable.kPNGDEC_InterlaceErrorFlag (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable.kPNGDEC_WidthErrorFlag (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S._pngdec_interrupt_enable.kPNGDEC_ZlibHeaderErrorFlag (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S._power_dcdc_mode (*C enum*), 806
drivers_RT_RT700_MIMXRT798S._power_dcdc_mode.kPower_DCDCMode_HP (*C enumerator*), 806
drivers_RT_RT700_MIMXRT798S._power_dcdc_mode.kPower_DCDCMode_LP (*C enumerator*), 806
drivers_RT_RT700_MIMXRT798S._power_dcdc_mode.kPower_DCDCMode_ULP (*C enumerator*), 806
drivers_RT_RT700_MIMXRT798S._power_hwwake_src (*C enum*), 803
drivers_RT_RT700_MIMXRT798S._power_hwwake_src.kPower_HWWakeSrcFlexio (*C enumerator*), 803
drivers_RT_RT700_MIMXRT798S._power_hwwake_src.kPower_HWWakeSrcLpFlexcomm (*C enumerator*), 803
drivers_RT_RT700_MIMXRT798S._power_hwwake_src.kPower_HWWakeSrcSai (*C enumerator*), 803
drivers_RT_RT700_MIMXRT798S._power_ldo_mode (*C enum*), 805
drivers_RT_RT700_MIMXRT798S._power_ldo_mode.kPower_LDOMode_Bypass (*C enumerator*), 806
drivers_RT_RT700_MIMXRT798S._power_ldo_mode.kPower_LDOMode_HP (*C enumerator*), 806

drivers_RT_RT700_MIMXRT798S._power_ldo_mode.kPower_LDOMode_LP (*C enumerator*), 806
drivers_RT_RT700_MIMXRT798S._power_lvd_voltage (*C union*), 818
drivers_RT_RT700_MIMXRT798S._power_lvd_voltage.VDD12 (*C struct*), 819
drivers_RT_RT700_MIMXRT798S._power_lvd_voltage.VDD12.lvl0 (*C var*), 819
drivers_RT_RT700_MIMXRT798S._power_lvd_voltage.VDD12.lvl1 (*C var*), 819
drivers_RT_RT700_MIMXRT798S._power_lvd_voltage.VDD12.lvl2 (*C var*), 819
drivers_RT_RT700_MIMXRT798S._power_lvd_voltage.VDD12.lvl3 (*C var*), 819
drivers_RT_RT700_MIMXRT798S._power_lvd_voltage.VDDN (*C struct*), 819
drivers_RT_RT700_MIMXRT798S._power_lvd_voltage.VDDN.lvl0 (*C var*), 819
drivers_RT_RT700_MIMXRT798S._power_lvd_voltage.VDDN.lvl1 (*C var*), 819
drivers_RT_RT700_MIMXRT798S._power_mode_config (*C enum*), 803
drivers_RT_RT700_MIMXRT798S._power_mode_config.kPower_DeepPowerDown (*C enumerator*), 803
drivers_RT_RT700_MIMXRT798S._power_mode_config.kPower_DeepSleep (*C enumerator*), 803
drivers_RT_RT700_MIMXRT798S._power_mode_config.kPower_DeepSleepRetention (*C enumerator*), 803
drivers_RT_RT700_MIMXRT798S._power_mode_config.kPower_FullDeepPowerDown (*C enumerator*), 803
drivers_RT_RT700_MIMXRT798S._power_mode_config.kPower_Sleep (*C enumerator*), 803
drivers_RT_RT700_MIMXRT798S._power_pad_vrange_val (*C enum*), 804
drivers_RT_RT700_MIMXRT798S._power_pad_vrange_val.kPadVol_171_198 (*C enumerator*), 805
drivers_RT_RT700_MIMXRT798S._power_pad_vrange_val.kPadVol_300_360 (*C enumerator*), 805
drivers_RT_RT700_MIMXRT798S._power_pad_vrange_val.kPadVol_Continuous (*C enumerator*), 805
drivers_RT_RT700_MIMXRT798S._power_regulator (*C enum*), 805
drivers_RT_RT700_MIMXRT798S._power_regulator_voltage (*C union*), 818
drivers_RT_RT700_MIMXRT798S._power_regulator_voltage.DCDC (*C struct*), 818
drivers_RT_RT700_MIMXRT798S._power_regulator_voltage.DCDC.vsel0 (*C var*), 818
drivers_RT_RT700_MIMXRT798S._power_regulator_voltage.DCDC.vsel1 (*C var*), 819
drivers_RT_RT700_MIMXRT798S._power_regulator_voltage.LDO (*C struct*), 819
drivers_RT_RT700_MIMXRT798S._power_regulator_voltage.LDO.vsel0 (*C var*), 819
drivers_RT_RT700_MIMXRT798S._power_regulator_voltage.LDO.vsel1 (*C var*), 819
drivers_RT_RT700_MIMXRT798S._power_regulator_voltage.LDO.vsel2 (*C var*), 819
drivers_RT_RT700_MIMXRT798S._power_regulator_voltage.LDO.vsel3 (*C var*), 819
drivers_RT_RT700_MIMXRT798S._power_regulator.kRegulator_DCDC (*C enumerator*), 805
drivers_RT_RT700_MIMXRT798S._power_regulator.kRegulator_Vdd1LDO (*C enumerator*), 805
drivers_RT_RT700_MIMXRT798S._power_regulator.kRegulator_Vdd2LDO (*C enumerator*), 805
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask (*C enum*), 803
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskAcmp0 (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskAdc0 (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskAll (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskEzhv (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskFlexio (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskGdet3 (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskLpi2c15 (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskLpspi14 (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskLpspi16 (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskMicfil (*C enumerator*), 804

drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskNic0 (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskNic1 (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskRtc (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskVgpu (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_shared_resource_mask.kPower_MaskXspi2 (*C enumerator*), 804
drivers_RT_RT700_MIMXRT798S._power_vdd_src (*C enum*), 806
drivers_RT_RT700_MIMXRT798S._power_vdd_src.kVddSrc_PMC (*C enumerator*), 806
drivers_RT_RT700_MIMXRT798S._power_vdd_src.kVddSrc_PMIC (*C enumerator*), 806
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source (*C enum*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceAll (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceCdog0 (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceCdog1 (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceCdog2 (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceCdog3 (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceCdog4 (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceCpu0 (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceCpu1 (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceIspAp (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceItrcSw (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourcePad (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceVddPor (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceWwdt0 (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceWwdt1 (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceWwdt2 (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._rstctl_reset_source.kRSTCTL_SourceWwdt3 (*C enumerator*), 829
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn (*C enum*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kACMP0_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kADC0_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kCPU1_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kCRC0_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kCTIMER0_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kCTIMER1_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kCTIMER2_RST_SHIFT_RSTn (*C enumera-*

tor), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kCTIMER3_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kCTIMER4_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kCTIMER5_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kCTIMER6_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kCTIMER7_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kDMA0_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kDMA1_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kDMA2_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kDMA3_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kELS_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kEZHV_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC0_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC1_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC2_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC3_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC4_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC5_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC6_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC7_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC8_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC9_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC10_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC11_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC12_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC13_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC17_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC18_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC19_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFC20_RST_SHIFT_RSTn (*C enumerator*),

834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFLEXIO0_RST_SHIFT_RSTn (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kFREQME_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kGPIO0_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kGPIO1_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kGPIO2_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kGPIO3_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kGPIO4_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kGPIO5_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kGPIO6_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kGPIO7_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kGPIO8_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kGPIO9_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kGPIO10_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kHIFI1_DEBUG_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kHIFI1_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kHIFI4_DEBUG_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kHIFI4_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kI3C0_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kI3C1_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kI3C2_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kI3C3_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kINPUTMUX0_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kINPUTMUX1_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kIOPCTL0_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kIOPCTL1_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kIOPCTL2_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kJPEGDEC_RST_SHIFT_RSTn (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kLCDIF_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kLPI2C15_RST_SHIFT_RSTn (*C enumera-*

tor), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kLPSPI14_RST_SHIFT_RSTn (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kLPSPI16_RST_SHIFT_RSTn (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kMIPI_DSI_CTRL_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kMRT0_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kMRT1_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kMU0_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kMU1_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kMU2_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kMU3_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kMU4_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kNPU0_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kPDM_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kPKC_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kPNGDEC_RST_SHIFT_RSTn (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kPVTS1_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSAFO_SGI_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSAI0_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSAI1_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSAI2_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSAI3_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSCT0_RST_SHIFT_RSTn (*C enumerator*), 831
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSDADC0_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSEMA420_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSEMA423_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSEMA424_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSYSPM0_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSYSPM1_RST_SHIFT_RSTn (*C enumerator*), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSYSPM2_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kSYSPM3_RST_SHIFT_RSTn (*C enumera-*

tor), 833
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kUSB0_RST_SHIFT_RSTn (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kUSB1_RST_SHIFT_RSTn (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kUSBPHY0_RST_SHIFT_RSTn (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kUSDHC0_RST_SHIFT_RSTn (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kUSDHC1_RST_SHIFT_RSTn (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kUTICK0_RST_SHIFT_RSTn (*C enumerator*), 832
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kUTICK1_RST_SHIFT_RSTn (*C enumerator*), 834
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kVGPU_RST_SHIFT_RSTn (*C enumerator*), 835
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kXSPI0_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kXSPI1_RST_SHIFT_RSTn (*C enumerator*), 830
drivers_RT_RT700_MIMXRT798S._RSTCTL_RSTn.kXSPI2_RST_SHIFT_RSTn (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S._sai_bclk_source (*C enum*), 860
drivers_RT_RT700_MIMXRT798S._sai_bclk_source.kSAI_BclkSourceBusclk (*C enumerator*), 860
drivers_RT_RT700_MIMXRT798S._sai_bclk_source.kSAI_BclkSourceMclkDiv (*C enumerator*), 860
drivers_RT_RT700_MIMXRT798S._sai_bclk_source.kSAI_BclkSourceMclkOption1 (*C enumerator*), 860
drivers_RT_RT700_MIMXRT798S._sai_bclk_source.kSAI_BclkSourceMclkOption2 (*C enumerator*), 860
drivers_RT_RT700_MIMXRT798S._sai_bclk_source.kSAI_BclkSourceMclkOption3 (*C enumerator*), 860
drivers_RT_RT700_MIMXRT798S._sai_bclk_source.kSAI_BclkSourceOtherSai0 (*C enumerator*), 860
drivers_RT_RT700_MIMXRT798S._sai_bclk_source.kSAI_BclkSourceOtherSai1 (*C enumerator*), 860
drivers_RT_RT700_MIMXRT798S._sai_bit_clock (*C struct*), 868
drivers_RT_RT700_MIMXRT798S._sai_bit_clock.bclkInputDelay (*C var*), 868
drivers_RT_RT700_MIMXRT798S._sai_bit_clock.bclkPolarity (*C var*), 869
drivers_RT_RT700_MIMXRT798S._sai_bit_clock.bclkSource (*C var*), 869
drivers_RT_RT700_MIMXRT798S._sai_bit_clock.bclkSrcSwap (*C var*), 868
drivers_RT_RT700_MIMXRT798S._sai_clock_polarity (*C enum*), 859
drivers_RT_RT700_MIMXRT798S._sai_clock_polarity.kSAI_PolarityActiveHigh (*C enumerator*), 859
drivers_RT_RT700_MIMXRT798S._sai_clock_polarity.kSAI_PolarityActiveLow (*C enumerator*), 859
drivers_RT_RT700_MIMXRT798S._sai_clock_polarity.kSAI_SampleOnFallingEdge (*C enumerator*), 860
drivers_RT_RT700_MIMXRT798S._sai_clock_polarity.kSAI_SampleOnRisingEdge (*C enumerator*), 860
drivers_RT_RT700_MIMXRT798S._sai_config (*C struct*), 866
drivers_RT_RT700_MIMXRT798S._sai_config.bclkSource (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_config.masterSlave (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_config.mclkOutputEnable (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_config.protocol (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_config.syncMode (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_data_order (*C enum*), 859
drivers_RT_RT700_MIMXRT798S._sai_data_order.kSAI_DataLSB (*C enumerator*), 859

drivers_RT_RT700_MIMXRT798S._sai_data_order.kSAI_DataMSB (*C enumerator*), 859
drivers_RT_RT700_MIMXRT798S._sai_data_pin_state (*C enum*), 863
drivers_RT_RT700_MIMXRT798S._sai_data_pin_state.kSAI_DataPinStateOutputZero (*C enumerator*), 863
drivers_RT_RT700_MIMXRT798S._sai_data_pin_state.kSAI_DataPinStateTriState (*C enumerator*), 863
drivers_RT_RT700_MIMXRT798S._sai_edma_interleave (*C enum*), 877
drivers_RT_RT700_MIMXRT798S._sai_edma_interleave.kSAI_EDMAInterleavePerChannelBlock (*C enumerator*), 877
drivers_RT_RT700_MIMXRT798S._sai_edma_interleave.kSAI_EDMAInterleavePerChannelSample (*C enumerator*), 877
drivers_RT_RT700_MIMXRT798S._sai_fifo (*C struct*), 868
drivers_RT_RT700_MIMXRT798S._sai_fifo_combine (*C enum*), 863
drivers_RT_RT700_MIMXRT798S._sai_fifo_combine.kSAI_FifoCombineDisabled (*C enumerator*), 864
drivers_RT_RT700_MIMXRT798S._sai_fifo_combine.kSAI_FifoCombineModeEnabledOnRead (*C enumerator*), 864
drivers_RT_RT700_MIMXRT798S._sai_fifo_combine.kSAI_FifoCombineModeEnabledOnReadWrite (*C enumerator*), 864
drivers_RT_RT700_MIMXRT798S._sai_fifo_combine.kSAI_FifoCombineModeEnabledOnWrite (*C enumerator*), 864
drivers_RT_RT700_MIMXRT798S._sai_fifo_combine.kSAI_RXFifoCombineModeEnabledOnRead (*C enumerator*), 864
drivers_RT_RT700_MIMXRT798S._sai_fifo_combine.kSAI_RxFifoCombineModeEnabledOnWrite (*C enumerator*), 864
drivers_RT_RT700_MIMXRT798S._sai_fifo_packing (*C enum*), 862
drivers_RT_RT700_MIMXRT798S._sai_fifo_packing.kSAI_FifoPacking8bit (*C enumerator*), 862
drivers_RT_RT700_MIMXRT798S._sai_fifo_packing.kSAI_FifoPacking16bit (*C enumerator*), 862
drivers_RT_RT700_MIMXRT798S._sai_fifo_packing.kSAI_FifoPackingDisabled (*C enumerator*), 862
drivers_RT_RT700_MIMXRT798S._sai_fifo.fifoCombine (*C var*), 868
drivers_RT_RT700_MIMXRT798S._sai_fifo.fifoContinueOneError (*C var*), 868
drivers_RT_RT700_MIMXRT798S._sai_fifo.fifoPacking (*C var*), 868
drivers_RT_RT700_MIMXRT798S._sai_fifo.fifoWatermark (*C var*), 868
drivers_RT_RT700_MIMXRT798S._sai_frame_sync (*C struct*), 869
drivers_RT_RT700_MIMXRT798S._sai_frame_sync_len (*C enum*), 864
drivers_RT_RT700_MIMXRT798S._sai_frame_sync_len.kSAI_FrameSyncLenOneBitClk (*C enumerator*), 864
drivers_RT_RT700_MIMXRT798S._sai_frame_sync_len.kSAI_FrameSyncLenPerWordWidth (*C enumerator*), 864
drivers_RT_RT700_MIMXRT798S._sai_frame_sync.frameSyncEarly (*C var*), 869
drivers_RT_RT700_MIMXRT798S._sai_frame_sync.frameSyncGenerateOnDemand (*C var*), 869
drivers_RT_RT700_MIMXRT798S._sai_frame_sync.frameSyncPolarity (*C var*), 869
drivers_RT_RT700_MIMXRT798S._sai_frame_sync.frameSyncWidth (*C var*), 869
drivers_RT_RT700_MIMXRT798S._sai_handle (*C struct*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.base (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.bitWidth (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.callback (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.channel (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.channelMask (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.channelNums (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.endChannel (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.queueDriver (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.queueUser (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.saiQueue (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.state (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.transferSize (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_handle.userData (*C var*), 871

drivers_RT_RT700_MIMXRT798S._sai_handle.watermark (*C var*), 872
drivers_RT_RT700_MIMXRT798S._sai_master_clock (*C struct*), 868
drivers_RT_RT700_MIMXRT798S._sai_master_clock.mclkHz (*C var*), 868
drivers_RT_RT700_MIMXRT798S._sai_master_clock.mclkOutputEnable (*C var*), 868
drivers_RT_RT700_MIMXRT798S._sai_master_clock.mclkSourceClkHz (*C var*), 868
drivers_RT_RT700_MIMXRT798S._sai_master_slave (*C enum*), 859
drivers_RT_RT700_MIMXRT798S._sai_master_slave.kSAI_Bclk_Master_FrameSync_Slave (*C enumerator*), 859
drivers_RT_RT700_MIMXRT798S._sai_master_slave.kSAI_Bclk_Slave_FrameSync_Master (*C enumerator*), 859
drivers_RT_RT700_MIMXRT798S._sai_master_slave.kSAI_Master (*C enumerator*), 859
drivers_RT_RT700_MIMXRT798S._sai_master_slave.kSAI_Slave (*C enumerator*), 859
drivers_RT_RT700_MIMXRT798S._sai_mono_stereo (*C enum*), 859
drivers_RT_RT700_MIMXRT798S._sai_mono_stereo.kSAI_MonoLeft (*C enumerator*), 859
drivers_RT_RT700_MIMXRT798S._sai_mono_stereo.kSAI_MonoRight (*C enumerator*), 859
drivers_RT_RT700_MIMXRT798S._sai_mono_stereo.kSAI_Stereo (*C enumerator*), 859
drivers_RT_RT700_MIMXRT798S._sai_protocol (*C enum*), 858
drivers_RT_RT700_MIMXRT798S._sai_protocol.kSAI_BusI2S (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S._sai_protocol.kSAI_BusLeftJustified (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S._sai_protocol.kSAI_BusPCMA (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S._sai_protocol.kSAI_BusPCMB (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S._sai_protocol.kSAI_BusRightJustified (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S._sai_reset_type (*C enum*), 862
drivers_RT_RT700_MIMXRT798S._sai_reset_type.kSAI_ResetAll (*C enumerator*), 862
drivers_RT_RT700_MIMXRT798S._sai_reset_type.kSAI_ResetTypeFIFO (*C enumerator*), 862
drivers_RT_RT700_MIMXRT798S._sai_reset_type.kSAI_ResetTypeSoftware (*C enumerator*), 862
drivers_RT_RT700_MIMXRT798S._sai_sample_rate (*C enum*), 862
drivers_RT_RT700_MIMXRT798S._sai_sample_rate.kSAI_SampleRate8KHz (*C enumerator*), 862
drivers_RT_RT700_MIMXRT798S._sai_sample_rate.kSAI_SampleRate12KHz (*C enumerator*), 862
drivers_RT_RT700_MIMXRT798S._sai_sample_rate.kSAI_SampleRate16KHz (*C enumerator*), 862
drivers_RT_RT700_MIMXRT798S._sai_sample_rate.kSAI_SampleRate24KHz (*C enumerator*), 862
drivers_RT_RT700_MIMXRT798S._sai_sample_rate.kSAI_SampleRate32KHz (*C enumerator*), 863
drivers_RT_RT700_MIMXRT798S._sai_sample_rate.kSAI_SampleRate48KHz (*C enumerator*), 863
drivers_RT_RT700_MIMXRT798S._sai_sample_rate.kSAI_SampleRate96KHz (*C enumerator*), 863
drivers_RT_RT700_MIMXRT798S._sai_sample_rate.kSAI_SampleRate192KHz (*C enumerator*), 863
drivers_RT_RT700_MIMXRT798S._sai_sample_rate.kSAI_SampleRate384KHz (*C enumerator*), 863
drivers_RT_RT700_MIMXRT798S._sai_sample_rate.kSAI_SampleRate11025Hz (*C enumerator*), 862
drivers_RT_RT700_MIMXRT798S._sai_sample_rate.kSAI_SampleRate22050Hz (*C enumerator*), 862
drivers_RT_RT700_MIMXRT798S._sai_sample_rate.kSAI_SampleRate44100Hz (*C enumerator*), 863
drivers_RT_RT700_MIMXRT798S._sai_serial_data (*C struct*), 869
drivers_RT_RT700_MIMXRT798S._sai_serial_data.dataFirstBitShifted (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_serial_data.dataMaskedWord (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_serial_data.dataMode (*C var*), 869
drivers_RT_RT700_MIMXRT798S._sai_serial_data.dataOrder (*C var*), 869
drivers_RT_RT700_MIMXRT798S._sai_serial_data.dataWord0Length (*C var*), 869
drivers_RT_RT700_MIMXRT798S._sai_serial_data.dataWordLength (*C var*), 869
drivers_RT_RT700_MIMXRT798S._sai_serial_data.dataWordNLength (*C var*), 869
drivers_RT_RT700_MIMXRT798S._sai_serial_data.dataWordNum (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_sync_mode (*C enum*), 860
drivers_RT_RT700_MIMXRT798S._sai_sync_mode.kSAI_ModeAsync (*C enumerator*), 860
drivers_RT_RT700_MIMXRT798S._sai_sync_mode.kSAI_ModeSync (*C enumerator*), 860
drivers_RT_RT700_MIMXRT798S._sai_sync_mode.kSAI_ModeSyncWithOtherRx (*C enumerator*), 860

drivers_RT_RT700_MIMXRT798S._sai_sync_mode.kSAI_ModeSyncWithOtherTx (*C enumerator*), 860
drivers_RT_RT700_MIMXRT798S._sai_transceiver (*C struct*), 870
drivers_RT_RT700_MIMXRT798S._sai_transceiver_type (*C enum*), 864
drivers_RT_RT700_MIMXRT798S._sai_transceiver_type.kSAI_Receiver (*C enumerator*), 864
drivers_RT_RT700_MIMXRT798S._sai_transceiver_type.kSAI_Transmitter (*C enumerator*), 864
drivers_RT_RT700_MIMXRT798S._sai_transceiver.bitClock (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_transceiver.channelMask (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_transceiver.channelNums (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_transceiver.endChannel (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_transceiver.fifo (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_transceiver.frameSync (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_transceiver.masterSlave (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_transceiver.serialData (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_transceiver.startChannel (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_transceiver.syncMode (*C var*), 870
drivers_RT_RT700_MIMXRT798S._sai_transfer (*C struct*), 870
drivers_RT_RT700_MIMXRT798S._sai_transfer_format (*C struct*), 867
drivers_RT_RT700_MIMXRT798S._sai_transfer_format.bitWidth (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_transfer_format.channel (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_transfer_format.channelMask (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_transfer_format.channelNums (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_transfer_format.endChannel (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_transfer_format.isFrameSyncCompact (*C var*), 868
drivers_RT_RT700_MIMXRT798S._sai_transfer_format.masterClockHz (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_transfer_format.protocol (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_transfer_format.sampleRate_Hz (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_transfer_format.stereo (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_transfer_format.watermark (*C var*), 867
drivers_RT_RT700_MIMXRT798S._sai_transfer.data (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_transfer.dataSize (*C var*), 871
drivers_RT_RT700_MIMXRT798S._sai_word_width (*C enum*), 863
drivers_RT_RT700_MIMXRT798S._sai_word_width.kSAI_WordWidth8bits (*C enumerator*), 863
drivers_RT_RT700_MIMXRT798S._sai_word_width.kSAI_WordWidth16bits (*C enumerator*), 863
drivers_RT_RT700_MIMXRT798S._sai_word_width.kSAI_WordWidth24bits (*C enumerator*), 863
drivers_RT_RT700_MIMXRT798S._sai_word_width.kSAI_WordWidth32bits (*C enumerator*), 863
drivers_RT_RT700_MIMXRT798S._sctimer_clock_mode (*C enum*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_clock_mode.kSCTIMER_Asynchronous_ClockMode (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_mode.kSCTIMER_Input_ClockMode (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_mode.kSCTIMER_Sampled_ClockMode (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_mode.kSCTIMER_System_ClockMode (*C enumerator*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select (*C enum*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_0 (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_1 (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_2 (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_3 (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_4 (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_5 (*C enumerator*), 900

drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_6 (*C enumerator*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_7 (*C enumerator*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_0 (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_1 (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_2 (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_3 (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_4 (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_5 (*C enumerator*), 900
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_6 (*C enumerator*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_7 (*C enumerator*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_complementary_pwm_handle (*C struct*), 906
drivers_RT_RT700_MIMXRT798S._sctimer_complementary_pwm_handle.deadTimeTicks (*C var*), 907
drivers_RT_RT700_MIMXRT798S._sctimer_complementary_pwm_handle.highFallEvent (*C var*), 907
drivers_RT_RT700_MIMXRT798S._sctimer_complementary_pwm_handle.highRiseEvent (*C var*), 907
drivers_RT_RT700_MIMXRT798S._sctimer_complementary_pwm_handle.lowRiseEvent (*C var*), 907
drivers_RT_RT700_MIMXRT798S._sctimer_complementary_pwm_handle.outHigh (*C var*), 907
drivers_RT_RT700_MIMXRT798S._sctimer_complementary_pwm_handle.outLow (*C var*), 907
drivers_RT_RT700_MIMXRT798S._sctimer_complementary_pwm_handle.periodEvent (*C var*), 906
drivers_RT_RT700_MIMXRT798S._sctimer_config (*C struct*), 907
drivers_RT_RT700_MIMXRT798S._sctimer_config.clockMode (*C var*), 907
drivers_RT_RT700_MIMXRT798S._sctimer_config.clockSelect (*C var*), 907
drivers_RT_RT700_MIMXRT798S._sctimer_config.enableBidirection_h (*C var*), 908
drivers_RT_RT700_MIMXRT798S._sctimer_config.enableBidirection_l (*C var*), 907
drivers_RT_RT700_MIMXRT798S._sctimer_config.enableCounterUnify (*C var*), 907
drivers_RT_RT700_MIMXRT798S._sctimer_config.inputsync (*C var*), 908
drivers_RT_RT700_MIMXRT798S._sctimer_config.outInitState (*C var*), 908
drivers_RT_RT700_MIMXRT798S._sctimer_config.prescale_h (*C var*), 908
drivers_RT_RT700_MIMXRT798S._sctimer_config.prescale_l (*C var*), 908
drivers_RT_RT700_MIMXRT798S._sctimer_conflict_resolution (*C enum*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_conflict_resolution.kSCTIMER_ResolveClear (*C enumerator*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_conflict_resolution.kSCTIMER_ResolveNone (*C enumerator*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_conflict_resolution.kSCTIMER_ResolveSet (*C enumerator*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_conflict_resolution.kSCTIMER_ResolveToggle (*C enumerator*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_counter (*C enum*), 897
drivers_RT_RT700_MIMXRT798S._sctimer_counter.kSCTIMER_Counter_H (*C enumerator*), 897
drivers_RT_RT700_MIMXRT798S._sctimer_counter.kSCTIMER_Counter_L (*C enumerator*), 897
drivers_RT_RT700_MIMXRT798S._sctimer_counter.kSCTIMER_Counter_U (*C enumerator*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_event (*C enum*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_event_active_direction (*C enum*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_event_active_direction.kSCTIMER_ActiveInCountDown

(*C enumerator*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_event_active_direction.kSCTIMER_ActiveInCountUp (*C enumerator*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_event_active_direction.kSCTIMER_ActiveIndependent (*C enumerator*), 901
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_InputFallAndMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_InputFallEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_InputFallOrMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_InputHighAndMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_InputHighEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_InputHighOrMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_InputLowAndMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_InputLowEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_InputLowOrMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_InputRiseAndMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_InputRiseEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_InputRiseOrMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_MatchEventOnly (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_OutputFallAndMatchEvent (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_OutputFallEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_OutputFallOrMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_OutputHighAndMatchEvent (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_OutputHighEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_OutputHighOrMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_OutputLowAndMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_OutputLowEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_OutputLowOrMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_OutputRiseAndMatchEvent (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_OutputRiseEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_event.kSCTIMER_OutputRiseOrMatchEvent (*C enumerator*), 902
drivers_RT_RT700_MIMXRT798S._sctimer_input (*C enum*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_input.kSCTIMER_Input_0 (*C enumerator*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_input.kSCTIMER_Input_1 (*C enumerator*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_input.kSCTIMER_Input_2 (*C enumerator*), 898

drivers_RT_RT700_MIMXRT798S._sctimer_input.kSCTIMER_Input_3 (*C enumerator*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_input.kSCTIMER_Input_4 (*C enumerator*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_input.kSCTIMER_Input_5 (*C enumerator*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_input.kSCTIMER_Input_6 (*C enumerator*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_input.kSCTIMER_Input_7 (*C enumerator*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable (*C enum*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event0InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event1InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event2InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event3InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event4InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event5InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event6InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event7InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event8InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event9InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event10InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event11InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_interrupt_enable.kSCTIMER_Event12InterruptEnable (*C enumerator*), 903
drivers_RT_RT700_MIMXRT798S._sctimer_out (*C enum*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_out.kSCTIMER_Out_0 (*C enumerator*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_out.kSCTIMER_Out_1 (*C enumerator*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_out.kSCTIMER_Out_2 (*C enumerator*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_out.kSCTIMER_Out_3 (*C enumerator*), 898
drivers_RT_RT700_MIMXRT798S._sctimer_out.kSCTIMER_Out_4 (*C enumerator*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_out.kSCTIMER_Out_5 (*C enumerator*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_out.kSCTIMER_Out_6 (*C enumerator*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_out.kSCTIMER_Out_7 (*C enumerator*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_out.kSCTIMER_Out_8 (*C enumerator*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_out.kSCTIMER_Out_9 (*C enumerator*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_pwm_level_select (*C enum*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_pwm_level_select.kSCTIMER_HighTrue (*C enumerator*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_pwm_level_select.kSCTIMER_LowTrue (*C enumerator*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_pwm_mode (*C enum*), 897
drivers_RT_RT700_MIMXRT798S._sctimer_pwm_mode.kSCTIMER_CenterAlignedPwm (*C enumerator*), 897
drivers_RT_RT700_MIMXRT798S._sctimer_pwm_mode.kSCTIMER_EdgeAlignedPwm (*C enumerator*), 897
drivers_RT_RT700_MIMXRT798S._sctimer_pwm_signal_param (*C struct*), 906
drivers_RT_RT700_MIMXRT798S._sctimer_pwm_signal_param.dutyCyclePercent (*C var*), 906
drivers_RT_RT700_MIMXRT798S._sctimer_pwm_signal_param.level (*C var*), 906
drivers_RT_RT700_MIMXRT798S._sctimer_pwm_signal_param.output (*C var*), 906
drivers_RT_RT700_MIMXRT798S._sctimer_pwm_update_mode (*C enum*), 899

drivers_RT_RT700_MIMXRT798S._sctimer_pwm_update_mode.kSCTIMER_UpdateImmediately (*C enumerator*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_pwm_update_mode.kSCTIMER_UpdateOnNextPeriod (*C enumerator*), 899
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags (*C enum*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_BusErrorHFlag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_BusErrorLFlag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event0Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event1Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event2Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event3Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event4Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event5Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event6Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event7Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event8Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event9Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event10Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event11Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sctimer_status_flags.kSCTIMER_Event12Flag (*C enumerator*), 904
drivers_RT_RT700_MIMXRT798S._sdadc_channel_config (*C struct*), 922
drivers_RT_RT700_MIMXRT798S._sdadc_channel_config.enableDacCompensation (*C var*), 923
drivers_RT_RT700_MIMXRT798S._sdadc_channel_config.enableDcFilter (*C var*), 923
drivers_RT_RT700_MIMXRT798S._sdadc_channel_config.enableDCLoop (*C var*), 923
drivers_RT_RT700_MIMXRT798S._sdadc_channel_config.enablePolarityInvert (*C var*), 923
drivers_RT_RT700_MIMXRT798S._sdadc_channel_config.mode (*C var*), 922
drivers_RT_RT700_MIMXRT798S._sdadc_channel_config.number (*C var*), 922
drivers_RT_RT700_MIMXRT798S._sdadc_channel_config.samplerate (*C var*), 923
drivers_RT_RT700_MIMXRT798S._sdadc_channel_config.type (*C var*), 922
drivers_RT_RT700_MIMXRT798S._sdadc_channel_config.volume (*C var*), 923
drivers_RT_RT700_MIMXRT798S._sdadc_channel_config.watermark (*C var*), 923
drivers_RT_RT700_MIMXRT798S._sdadc_channel_group (*C struct*), 923
drivers_RT_RT700_MIMXRT798S._sdadc_channel_group.number (*C var*), 924
drivers_RT_RT700_MIMXRT798S._sdadc_channel_group.type (*C var*), 924
drivers_RT_RT700_MIMXRT798S._sdadc_channel_mode (*C enum*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_channel_mode.kSDADC_Differential_Mode (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_channel_mode.kSDADC_SingleEnd_Mode (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_channel_number (*C enum*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_channel_number.kSDADC_Channel0 (*C enumerator*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_channel_number.kSDADC_Channel1 (*C enumerator*),

918
drivers_RT_RT700_MIMXRT798S._sdadc_channel_number.kSDADC_Channel2 (*C enumerator*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_channel_number.kSDADC_Channel3 (*C enumerator*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_channel_type (*C enum*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_channel_type.kSDADC_Channel_NandPSide_Type (*C enumerator*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_channel_type.kSDADC_Channel_NSide_Type (*C enumerator*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_channel_type.kSDADC_Channel_PSide_Type (*C enumerator*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_config (*C struct*), 923
drivers_RT_RT700_MIMXRT798S._sdadc_config.channelConfig (*C var*), 923
drivers_RT_RT700_MIMXRT798S._sdadc_config.channelCount (*C var*), 923
drivers_RT_RT700_MIMXRT798S._sdadc_decimator_sample_rate (*C enum*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_decimator_sample_rate.kSDADC_DecimatorSampleRate16kHz (*C enumerator*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_decimator_sample_rate.kSDADC_DecimatorSampleRate32kHz (*C enumerator*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_decimator_sample_rate.kSDADC_DecimatorSampleRate48kHz (*C enumerator*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_enable (*C enum*), 915
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_enable.kSDADC_Channel0_NSide_FifoFullIntEnable (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_enable.kSDADC_Channel0_PSide_FifoFullIntEnable (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_enable.kSDADC_Channel1_NSide_FifoFullIntEnable (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_enable.kSDADC_Channel1_PSide_FifoFullIntEnable (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_enable.kSDADC_Channel2_NSide_FifoFullIntEnable (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_enable.kSDADC_Channel2_PSide_FifoFullIntEnable (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_enable.kSDADC_Channel3_NSide_FifoFullIntEnable (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_enable.kSDADC_Channel3_PSide_FifoFullIntEnable (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_status_flag (*C enum*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_status_flag.kSDADC_Channel0_NSide_FifoFullIntStatusFlag (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_status_flag.kSDADC_Channel0_PSide_FifoFullIntStatusFlag (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_status_flag.kSDADC_Channel1_NSide_FifoFullIntStatusFlag (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_status_flag.kSDADC_Channel1_PSide_FifoFullIntStatusFlag (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_status_flag.kSDADC_Channel2_NSide_FifoFullIntStatusFlag (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_status_flag.kSDADC_Channel2_PSide_FifoFullIntStatusFlag (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_status_flag.kSDADC_Channel3_NSide_FifoFullIntStatusFlag (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_int_status_flag.kSDADC_Channel3_PSide_FifoFullIntStatusFlag (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_reset_operation (*C enum*), 919
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_reset_operation.kSDADC_FifoReadReset (*C enu-*

merator), 919
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_reset_operation.kSDADC_FifoWriteAndReadReset (*C enumerator*), 919
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_reset_operation.kSDADC_FifoWriteReset (*C enumerator*), 919
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_enable (*C enum*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_enable.kSDADC_Channel0_NSide_FifoWatermarkEnab (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_enable.kSDADC_Channel0_PSide_FifoWatermarkEnab (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_enable.kSDADC_Channel1_NSide_FifoWatermarkEnab (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_enable.kSDADC_Channel1_PSide_FifoWatermarkEnab (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_enable.kSDADC_Channel2_NSide_FifoWatermarkEnab (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_enable.kSDADC_Channel2_PSide_FifoWatermarkEnab (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_enable.kSDADC_Channel3_NSide_FifoWatermarkEnab (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_enable.kSDADC_Channel3_PSide_FifoWatermarkEnab (*C enumerator*), 916
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_status_flag (*C enum*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_status_flag.kSDADC_Channel0_NSide_FifoWatermark (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_status_flag.kSDADC_Channel0_PSide_FifoWatermarkS (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_status_flag.kSDADC_Channel1_NSide_FifoWatermark (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_status_flag.kSDADC_Channel1_PSide_FifoWatermarkS (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_status_flag.kSDADC_Channel2_NSide_FifoWatermark (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_status_flag.kSDADC_Channel2_PSide_FifoWatermarkS (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_status_flag.kSDADC_Channel3_NSide_FifoWatermark (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_fifo_watermark_status_flag.kSDADC_Channel3_PSide_FifoWatermarkS (*C enumerator*), 917
drivers_RT_RT700_MIMXRT798S._sdadc_ldo_drive_capability (*C enum*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_ldo_drive_capability.kSDADC_LdoHighPower (*C enumerator*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_ldo_drive_capability.kSDADC_LdoLowPower (*C enumerator*), 918
drivers_RT_RT700_MIMXRT798S._sdadc_ref_mode (*C enum*), 919
drivers_RT_RT700_MIMXRT798S._sdadc_ref_mode.kSDADC_RefFastChargeMode (*C enumerator*), 919
drivers_RT_RT700_MIMXRT798S._sdadc_ref_mode.kSDADC_RefLowNoiseMode (*C enumerator*), 919
drivers_RT_RT700_MIMXRT798S._sdadc_ref_mode.kSDADC_RefLowPowerMode (*C enumerator*), 919
drivers_RT_RT700_MIMXRT798S._sdadc_ref_mode.kSDADC_RefRetainMode (*C enumerator*), 919
drivers_RT_RT700_MIMXRT798S._sdadc_sample_rate (*C struct*), 922
drivers_RT_RT700_MIMXRT798S._sdadc_sample_rate.nSideSampleRate (*C var*), 922
drivers_RT_RT700_MIMXRT798S._sdadc_sample_rate.pSideSampleRate (*C var*), 922
drivers_RT_RT700_MIMXRT798S._sdadc_volume (*C struct*), 921
drivers_RT_RT700_MIMXRT798S._sdadc_volume.nSideVolume (*C var*), 922

drivers_RT_RT700_MIMXRT798S._sdadc_volume.pSideVolume (*C var*), 922
drivers_RT_RT700_MIMXRT798S._sdadc_watermark (*C struct*), 922
drivers_RT_RT700_MIMXRT798S._sdadc_watermark.nSideWatermark (*C var*), 922
drivers_RT_RT700_MIMXRT798S._sdadc_watermark.pSideWatermark (*C var*), 922
drivers_RT_RT700_MIMXRT798S._SDK_AtomicLocalCompareAndSet (*C function*), 556
drivers_RT_RT700_MIMXRT798S._SDK_AtomicTestAndSet (*C function*), 556
drivers_RT_RT700_MIMXRT798S._sectionType (*C enum*), 879
drivers_RT_RT700_MIMXRT798S._sectionType.kSectionDataRange (*C enumerator*), 879
drivers_RT_RT700_MIMXRT798S._sectionType.kSectionDDRConfig (*C enumerator*), 879
drivers_RT_RT700_MIMXRT798S._sectionType.kSectionDiffUpdate (*C enumerator*), 879
drivers_RT_RT700_MIMXRT798S._sectionType.kSectionNone (*C enumerator*), 879
drivers_RT_RT700_MIMXRT798S._sectionType.kSectionRegister (*C enumerator*), 879
drivers_RT_RT700_MIMXRT798S._sema42_gate_status (*C enum*), 924
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc0 (*C enumerator*), 924
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc1 (*C enumerator*), 924
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc2 (*C enumerator*), 924
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc3 (*C enumerator*), 924
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc4 (*C enumerator*), 924
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc5 (*C enumerator*), 924
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc6 (*C enumerator*), 925
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc7 (*C enumerator*), 925
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc8 (*C enumerator*), 925
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc9 (*C enumerator*), 925
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc10 (*C enumerator*), 925
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc11 (*C enumerator*), 925
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc12 (*C enumerator*), 925
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc13 (*C enumerator*), 925
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_LockedByProc14 (*C enumerator*), 925
drivers_RT_RT700_MIMXRT798S._sema42_gate_status.kSEMA42_Unlocked (*C enumerator*), 924
drivers_RT_RT700_MIMXRT798S._status_groups (*C enum*), 544
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_AON_COMMON (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_AON_POWER (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_ApplicationRangeStart (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_ASRC (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_AUDMIX (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_BMA (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_BUTTON (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_CAAM (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_CLIF (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_CODEC (*C enumerator*), 551

drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_COMMON_TASK (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_CSI (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_CSS_PKC (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_DCP (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_DebugConsole (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_DMA (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_DMAMGR (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_DMIC (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_DSPI (*C enumerator*), 544
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_ECSPI (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_EDMA (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_ELA_CSEC (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_ELE (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_ELEMU (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_ELS_PKC (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_ENDAT3 (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_ENET (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_ENET_QOS (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_ESAI (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_EVENT_CTRL (*C enumerator*), 553
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_EXTERN_EEPROM (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLASH (*C enumerator*), 544
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLASHIAP (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLEXCAN (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLEXCOMM_I2C (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLEXIO_A_FORMAT (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLEXIO_CAMERA (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLEXIO_I2C (*C enumerator*), 544
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLEXIO_I2S (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLEXIO_MCULCD (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLEXIO_SPI (*C enumerator*), 544
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLEXIO_T_FORMAT (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLEXIO_UART (*C enumerator*), 544
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_FLEXSPI (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_Generic (*C enumerator*), 544
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_GLIKEY (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HAL_ADC_SENSOR (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HAL_FLASH (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HAL_GPIO (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HAL_I2C (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HAL_I2S (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HAL_PWM (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HAL_RNG (*C enumerator*), 550

drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HAL_SPI (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HAL_TIMER (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HAL_UART (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HASHCRYPT (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HIPERFACE (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_HOSTIF (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_I2C (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_I2S (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_I3C (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_I3CBUS (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_IAP (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_ICS (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_IPED (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_IUART (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_JPEGDEC (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LED (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LIST (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LMEM (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LOG (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LPC_I2C (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LPC_I2C_1 (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LPC_MINISPI (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LPC_QSPI (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LPC_SPI (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LPC_SPI_SSP (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LPC_USART (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LPI2C (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LPSCI (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LPSPI (*C enumerator*), 544
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LPUART (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_LTC (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_MCAN (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_MCG (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_MECC (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_MEM_MANAGER (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_MIPI_DSI (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_MMDC (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_MSCAN (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_MSG (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_NETC (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_NOTIFIER (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_NPX (*C enumerator*), 552
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_OSA (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_OTFAD (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_OTP (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_PDM (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_PHY (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_PNGDEC (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_POWER (*C enumerator*), 546

drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_POWER_MANAGER (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_PUF (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_QSCI (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_QSPI (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_QUEUEDSPI (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SAI (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SCG (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SDHC (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SDIF (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SDIOSLV (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SDK_FLEXSPINOR (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SDK_OCOTP (*C enumerator*), 551
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SDMA (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SDMMC (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SDRAMC (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SDSPI (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SEMA42 (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SEMC (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SERIALMANAGER (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SFA (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SHELL (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SMARTCARD (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SPC (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SPDIF (*C enumerator*), 548
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SPI (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_SPIFI (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_TIMERMANAGER (*C enumerator*), 550
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_TOUCH_PANEL (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_TRGMUX (*C enumerator*), 546
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_UART (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_USDHC (*C enumerator*), 547
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_VBAT (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_XRDC (*C enumerator*), 545
drivers_RT_RT700_MIMXRT798S._status_groups.kStatusGroup_XSPI (*C enumerator*), 549
drivers_RT_RT700_MIMXRT798S._syspm_event (*C enum*), 928
drivers_RT_RT700_MIMXRT798S._syspm_event.kSYSPM_Event1 (*C enumerator*), 928
drivers_RT_RT700_MIMXRT798S._syspm_event.kSYSPM_Event2 (*C enumerator*), 928
drivers_RT_RT700_MIMXRT798S._syspm_event.kSYSPM_Event3 (*C enumerator*), 928
drivers_RT_RT700_MIMXRT798S._syspm_mode (*C enum*), 928
drivers_RT_RT700_MIMXRT798S._syspm_mode.kSYSPM_BothMode (*C enumerator*), 928
drivers_RT_RT700_MIMXRT798S._syspm_mode.kSYSPM_PrivilegedMode (*C enumerator*), 928
drivers_RT_RT700_MIMXRT798S._syspm_mode.kSYSPM_UserMode (*C enumerator*), 928
drivers_RT_RT700_MIMXRT798S._syspm_monitor (*C enum*), 928
drivers_RT_RT700_MIMXRT798S._syspm_monitor.kSYSPM_Monitor0 (*C enumerator*), 928
drivers_RT_RT700_MIMXRT798S._syspm_startstop_control (*C enum*), 928
drivers_RT_RT700_MIMXRT798S._syspm_startstop_control.KSYSPM_EnableTraceControl (*C enumerator*), 929
drivers_RT_RT700_MIMXRT798S._syspm_startstop_control.kSYSPM_GlobalStart (*C enumerator*), 929

drivers_RT_RT700_MIMXRT798S._syspm_startstop_control.kSYSPM_GlobalStop (*C enumerator*), 929
drivers_RT_RT700_MIMXRT798S._syspm_startstop_control.kSYSPM_Idle (*C enumerator*), 929
drivers_RT_RT700_MIMXRT798S._syspm_startstop_control.kSYSPM_LocalStart (*C enumerator*), 929
drivers_RT_RT700_MIMXRT798S._syspm_startstop_control.kSYSPM_LocalStop (*C enumerator*), 929
drivers_RT_RT700_MIMXRT798S._trng_clock_mode (*C enum*), 933
drivers_RT_RT700_MIMXRT798S._trng_clock_mode.kTRNG_ClockModeRingOscillator (*C enumerator*), 933
drivers_RT_RT700_MIMXRT798S._trng_clock_mode.kTRNG_ClockModeSystem (*C enumerator*), 933
drivers_RT_RT700_MIMXRT798S._trng_ring_osc_div (*C enum*), 933
drivers_RT_RT700_MIMXRT798S._trng_ring_osc_div.kTRNG_RingOscDiv0 (*C enumerator*), 933
drivers_RT_RT700_MIMXRT798S._trng_ring_osc_div.kTRNG_RingOscDiv2 (*C enumerator*), 934
drivers_RT_RT700_MIMXRT798S._trng_ring_osc_div.kTRNG_RingOscDiv4 (*C enumerator*), 934
drivers_RT_RT700_MIMXRT798S._trng_ring_osc_div.kTRNG_RingOscDiv8 (*C enumerator*), 934
drivers_RT_RT700_MIMXRT798S._trng_sample_mode (*C enum*), 933
drivers_RT_RT700_MIMXRT798S._trng_sample_mode.kTRNG_SampleModeRaw (*C enumerator*), 933
drivers_RT_RT700_MIMXRT798S._trng_sample_mode.kTRNG_SampleModeVonNeumann (*C enumerator*), 933
drivers_RT_RT700_MIMXRT798S._trng_sample_mode.kTRNG_SampleModeVonNeumannRaw (*C enumerator*), 933
drivers_RT_RT700_MIMXRT798S._trng_statistical_check_limit (*C struct*), 935
drivers_RT_RT700_MIMXRT798S._trng_statistical_check_limit.maximum (*C var*), 935
drivers_RT_RT700_MIMXRT798S._trng_statistical_check_limit.minimum (*C var*), 935
drivers_RT_RT700_MIMXRT798S._trng_user_config (*C struct*), 935
drivers_RT_RT700_MIMXRT798S._trng_user_config.clockMode (*C var*), 936
drivers_RT_RT700_MIMXRT798S._trng_user_config.entropyDelay (*C var*), 936
drivers_RT_RT700_MIMXRT798S._trng_user_config.frequencyCountLimit (*C var*), 937
drivers_RT_RT700_MIMXRT798S._trng_user_config.lock (*C var*), 936
drivers_RT_RT700_MIMXRT798S._trng_user_config.longRunMaxLimit (*C var*), 936
drivers_RT_RT700_MIMXRT798S._trng_user_config.monobitLimit (*C var*), 936
drivers_RT_RT700_MIMXRT798S._trng_user_config.oscillatorMode (*C var*), 936
drivers_RT_RT700_MIMXRT798S._trng_user_config.pokerLimit (*C var*), 937
drivers_RT_RT700_MIMXRT798S._trng_user_config.retryCount (*C var*), 936
drivers_RT_RT700_MIMXRT798S._trng_user_config.ringOsc2Div (*C var*), 936
drivers_RT_RT700_MIMXRT798S._trng_user_config.ringOscDiv (*C var*), 936
drivers_RT_RT700_MIMXRT798S._trng_user_config.runBit1Limit (*C var*), 937
drivers_RT_RT700_MIMXRT798S._trng_user_config.runBit2Limit (*C var*), 937
drivers_RT_RT700_MIMXRT798S._trng_user_config.runBit3Limit (*C var*), 937
drivers_RT_RT700_MIMXRT798S._trng_user_config.runBit4Limit (*C var*), 937
drivers_RT_RT700_MIMXRT798S._trng_user_config.runBit5Limit (*C var*), 937
drivers_RT_RT700_MIMXRT798S._trng_user_config.runBit6PlusLimit (*C var*), 937
drivers_RT_RT700_MIMXRT798S._trng_user_config.sampleMode (*C var*), 936
drivers_RT_RT700_MIMXRT798S._trng_user_config.sampleSize (*C var*), 936
drivers_RT_RT700_MIMXRT798S._trng_user_config.sparseBitLimit (*C var*), 936
drivers_RT_RT700_MIMXRT798S._usdhc_adma2_descriptor (*C struct*), 966
drivers_RT_RT700_MIMXRT798S._usdhc_adma2_descriptor.address (*C var*), 966
drivers_RT_RT700_MIMXRT798S._usdhc_adma2_descriptor.attribute (*C var*), 966
drivers_RT_RT700_MIMXRT798S._usdhc_adma_config (*C struct*), 968
drivers_RT_RT700_MIMXRT798S._usdhc_adma_config.admaTable (*C var*), 968
drivers_RT_RT700_MIMXRT798S._usdhc_adma_config.admaTableWords (*C var*), 968
drivers_RT_RT700_MIMXRT798S._usdhc_adma_config.dmaMode (*C var*), 968
drivers_RT_RT700_MIMXRT798S._usdhc_boot_config (*C struct*), 966
drivers_RT_RT700_MIMXRT798S._usdhc_boot_config.ackTimeoutCount (*C var*), 967
drivers_RT_RT700_MIMXRT798S._usdhc_boot_config.blockCount (*C var*), 967

drivers_RT_RT700_MIMXRT798S._usdhc_boot_config.blockSize (*C var*), 967
drivers_RT_RT700_MIMXRT798S._usdhc_boot_config.bootMode (*C var*), 967
drivers_RT_RT700_MIMXRT798S._usdhc_boot_config.enableAutoStopAtBlockGap (*C var*), 967
drivers_RT_RT700_MIMXRT798S._usdhc_boot_config.enableBootAck (*C var*), 967
drivers_RT_RT700_MIMXRT798S._usdhc_boot_mode (*C enum*), 959
drivers_RT_RT700_MIMXRT798S._usdhc_boot_mode.kUSDHC_BootModeAlternative (*C enumerator*), 959
drivers_RT_RT700_MIMXRT798S._usdhc_boot_mode.kUSDHC_BootModeNormal (*C enumerator*), 959
drivers_RT_RT700_MIMXRT798S._usdhc_burst_len (*C enum*), 962
drivers_RT_RT700_MIMXRT798S._usdhc_burst_len.kUSDHC_EnBurstLenForINCR (*C enumerator*), 962
drivers_RT_RT700_MIMXRT798S._usdhc_burst_len.kUSDHC_EnBurstLenForINCR4816 (*C enumerator*), 962
drivers_RT_RT700_MIMXRT798S._usdhc_burst_len.kUSDHC_EnBurstLenForINCR4816WRAP (*C enumerator*), 962
drivers_RT_RT700_MIMXRT798S._usdhc_capability (*C struct*), 966
drivers_RT_RT700_MIMXRT798S._usdhc_capability.flags (*C var*), 966
drivers_RT_RT700_MIMXRT798S._usdhc_capability.maxBlockCount (*C var*), 966
drivers_RT_RT700_MIMXRT798S._usdhc_capability.maxBlockLength (*C var*), 966
drivers_RT_RT700_MIMXRT798S._usdhc_capability.mmcVersion (*C var*), 966
drivers_RT_RT700_MIMXRT798S._usdhc_capability.sdVersion (*C var*), 966
drivers_RT_RT700_MIMXRT798S._usdhc_card_command_type (*C enum*), 959
drivers_RT_RT700_MIMXRT798S._usdhc_card_command_type.kCARD_CommandTypeAbort (*C enumerator*), 959
drivers_RT_RT700_MIMXRT798S._usdhc_card_command_type.kCARD_CommandTypeEmpty (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S._usdhc_card_command_type.kCARD_CommandTypeNormal (*C enumerator*), 959
drivers_RT_RT700_MIMXRT798S._usdhc_card_command_type.kCARD_CommandTypeResume (*C enumerator*), 959
drivers_RT_RT700_MIMXRT798S._usdhc_card_command_type.kCARD_CommandTypeSuspend (*C enumerator*), 959
drivers_RT_RT700_MIMXRT798S._usdhc_card_response_type (*C enum*), 960
drivers_RT_RT700_MIMXRT798S._usdhc_card_response_type.kCARD_ResponseTypeNone (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S._usdhc_card_response_type.kCARD_ResponseTypeR1 (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S._usdhc_card_response_type.kCARD_ResponseTypeR1b (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S._usdhc_card_response_type.kCARD_ResponseTypeR2 (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S._usdhc_card_response_type.kCARD_ResponseTypeR3 (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S._usdhc_card_response_type.kCARD_ResponseTypeR4 (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S._usdhc_card_response_type.kCARD_ResponseTypeR5 (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S._usdhc_card_response_type.kCARD_ResponseTypeR5b (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S._usdhc_card_response_type.kCARD_ResponseTypeR6 (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S._usdhc_card_response_type.kCARD_ResponseTypeR7 (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S._usdhc_command (*C struct*), 967
drivers_RT_RT700_MIMXRT798S._usdhc_command.argument (*C var*), 968
drivers_RT_RT700_MIMXRT798S._usdhc_command.flags (*C var*), 968
drivers_RT_RT700_MIMXRT798S._usdhc_command.index (*C var*), 968

drivers_RT_RT700_MIMXRT798S._usdhc_command.response (*C var*), 968
drivers_RT_RT700_MIMXRT798S._usdhc_command.responseErrorFlags (*C var*), 968
drivers_RT_RT700_MIMXRT798S._usdhc_command.responseType (*C var*), 968
drivers_RT_RT700_MIMXRT798S._usdhc_command.type (*C var*), 968
drivers_RT_RT700_MIMXRT798S._usdhc_config (*C struct*), 967
drivers_RT_RT700_MIMXRT798S._usdhc_config.dataTimeout (*C var*), 967
drivers_RT_RT700_MIMXRT798S._usdhc_config.endianMode (*C var*), 967
drivers_RT_RT700_MIMXRT798S._usdhc_config.readWatermarkLevel (*C var*), 967
drivers_RT_RT700_MIMXRT798S._usdhc_config.writeWatermarkLevel (*C var*), 967
drivers_RT_RT700_MIMXRT798S._usdhc_data (*C struct*), 969
drivers_RT_RT700_MIMXRT798S._usdhc_data_bus_width (*C enum*), 958
drivers_RT_RT700_MIMXRT798S._usdhc_data_bus_width.kUSDHC_DataBusWidth1Bit (*C enumerator*), 958
drivers_RT_RT700_MIMXRT798S._usdhc_data_bus_width.kUSDHC_DataBusWidth4Bit (*C enumerator*), 958
drivers_RT_RT700_MIMXRT798S._usdhc_data_bus_width.kUSDHC_DataBusWidth8Bit (*C enumerator*), 958
drivers_RT_RT700_MIMXRT798S._usdhc_data.blockCount (*C var*), 970
drivers_RT_RT700_MIMXRT798S._usdhc_data.blockSize (*C var*), 970
drivers_RT_RT700_MIMXRT798S._usdhc_data.dataType (*C var*), 970
drivers_RT_RT700_MIMXRT798S._usdhc_data.enableAutoCommand12 (*C var*), 969
drivers_RT_RT700_MIMXRT798S._usdhc_data.enableAutoCommand23 (*C var*), 969
drivers_RT_RT700_MIMXRT798S._usdhc_data.enableIgnoreError (*C var*), 970
drivers_RT_RT700_MIMXRT798S._usdhc_data.rxData (*C var*), 970
drivers_RT_RT700_MIMXRT798S._usdhc_data.txData (*C var*), 970
drivers_RT_RT700_MIMXRT798S._usdhc_dma_mode (*C enum*), 958
drivers_RT_RT700_MIMXRT798S._usdhc_dma_mode.kUSDHC_DmaModeAdma1 (*C enumerator*), 958
drivers_RT_RT700_MIMXRT798S._usdhc_dma_mode.kUSDHC_DmaModeAdma2 (*C enumerator*), 958
drivers_RT_RT700_MIMXRT798S._usdhc_dma_mode.kUSDHC_DmaModeSimple (*C enumerator*), 958
drivers_RT_RT700_MIMXRT798S._usdhc_dma_mode.kUSDHC_ExternalDMA (*C enumerator*), 959
drivers_RT_RT700_MIMXRT798S._usdhc_endian_mode (*C enum*), 958
drivers_RT_RT700_MIMXRT798S._usdhc_endian_mode.kUSDHC_EndianModeBig (*C enumerator*), 958
drivers_RT_RT700_MIMXRT798S._usdhc_endian_mode.kUSDHC_EndianModeHalfWordBig (*C enumerator*), 958
drivers_RT_RT700_MIMXRT798S._usdhc_endian_mode.kUSDHC_EndianModeLittle (*C enumerator*), 958
drivers_RT_RT700_MIMXRT798S._usdhc_handle (*C struct*), 971
drivers_RT_RT700_MIMXRT798S._usdhc_handle.callback (*C var*), 971
drivers_RT_RT700_MIMXRT798S._usdhc_handle.command (*C var*), 971
drivers_RT_RT700_MIMXRT798S._usdhc_handle.data (*C var*), 971
drivers_RT_RT700_MIMXRT798S._usdhc_handle.enDMA (*C var*), 971
drivers_RT_RT700_MIMXRT798S._usdhc_handle.transferredWords (*C var*), 971
drivers_RT_RT700_MIMXRT798S._usdhc_handle.userData (*C var*), 971
drivers_RT_RT700_MIMXRT798S._usdhc_host (*C struct*), 971
drivers_RT_RT700_MIMXRT798S._usdhc_host.base (*C var*), 971
drivers_RT_RT700_MIMXRT798S._usdhc_host.capability (*C var*), 972
drivers_RT_RT700_MIMXRT798S._usdhc_host.config (*C var*), 972
drivers_RT_RT700_MIMXRT798S._usdhc_host.sourceClock_Hz (*C var*), 971
drivers_RT_RT700_MIMXRT798S._usdhc_host.transfer (*C var*), 972
drivers_RT_RT700_MIMXRT798S._usdhc_scatter_gather_data (*C struct*), 968
drivers_RT_RT700_MIMXRT798S._usdhc_scatter_gather_data_list (*C struct*), 968
drivers_RT_RT700_MIMXRT798S._usdhc_scatter_gather_data.blockSize (*C var*), 969
drivers_RT_RT700_MIMXRT798S._usdhc_scatter_gather_data.dataDirection (*C var*), 969

drivers_RT_RT700_MIMXRT798S._usdhc_scatter_gather_data.dataType (C var), 969
drivers_RT_RT700_MIMXRT798S._usdhc_scatter_gather_data.enableAutoCommand12 (C var), 969
drivers_RT_RT700_MIMXRT798S._usdhc_scatter_gather_data.enableAutoCommand23 (C var), 969
drivers_RT_RT700_MIMXRT798S._usdhc_scatter_gather_data.enableIgnoreError (C var), 969
drivers_RT_RT700_MIMXRT798S._usdhc_scatter_gather_data.sgData (C var), 969
drivers_RT_RT700_MIMXRT798S._usdhc_scatter_gather_transfer (C struct), 969
drivers_RT_RT700_MIMXRT798S._usdhc_scatter_gather_transfer.command (C var), 969
drivers_RT_RT700_MIMXRT798S._usdhc_scatter_gather_transfer.data (C var), 969
drivers_RT_RT700_MIMXRT798S._usdhc_transfer (C struct), 970
drivers_RT_RT700_MIMXRT798S._usdhc_transfer_callback (C struct), 970
drivers_RT_RT700_MIMXRT798S._usdhc_transfer_callback.BlockGap (C var), 970
drivers_RT_RT700_MIMXRT798S._usdhc_transfer_callback.CardInserted (C var), 970
drivers_RT_RT700_MIMXRT798S._usdhc_transfer_callback.CardRemoved (C var), 970
drivers_RT_RT700_MIMXRT798S._usdhc_transfer_callback.ReTuning (C var), 971
drivers_RT_RT700_MIMXRT798S._usdhc_transfer_callback.SdioInterrupt (C var), 970
drivers_RT_RT700_MIMXRT798S._usdhc_transfer_callback.TransferComplete (C var), 971
drivers_RT_RT700_MIMXRT798S._usdhc_transfer_direction (C enum), 958
drivers_RT_RT700_MIMXRT798S._usdhc_transfer_direction.kUSDHC_TransferDirectionReceive (C enumerator), 958
drivers_RT_RT700_MIMXRT798S._usdhc_transfer_direction.kUSDHC_TransferDirectionSend (C enumerator), 958
drivers_RT_RT700_MIMXRT798S._usdhc_transfer.command (C var), 970
drivers_RT_RT700_MIMXRT798S._usdhc_transfer.data (C var), 970
drivers_RT_RT700_MIMXRT798S._user_app_boot_invoke_option (C struct), 842
drivers_RT_RT700_MIMXRT798S._user_app_boot_invoke_option.option (C union), 842
drivers_RT_RT700_MIMXRT798S._user_app_boot_invoke_option.option.B (C struct), 842
drivers_RT_RT700_MIMXRT798S._user_app_boot_invoke_option.option.B (C var), 842
drivers_RT_RT700_MIMXRT798S._user_app_boot_invoke_option.option.U (C var), 842
drivers_RT_RT700_MIMXRT798S._utick_mode (C enum), 973
drivers_RT_RT700_MIMXRT798S._utick_mode.kUTICK_Onetime (C enumerator), 973
drivers_RT_RT700_MIMXRT798S._utick_mode.kUTICK_Repeat (C enumerator), 973
drivers_RT_RT700_MIMXRT798S._wwdt_config (C struct), 976
drivers_RT_RT700_MIMXRT798S._wwdt_config.clockFreq_Hz (C var), 977
drivers_RT_RT700_MIMXRT798S._wwdt_config.enableWatchdogProtect (C var), 977
drivers_RT_RT700_MIMXRT798S._wwdt_config.enableWatchdogReset (C var), 976
drivers_RT_RT700_MIMXRT798S._wwdt_config.enableWwdt (C var), 976
drivers_RT_RT700_MIMXRT798S._wwdt_config.timeoutValue (C var), 977
drivers_RT_RT700_MIMXRT798S._wwdt_config.warningValue (C var), 977
drivers_RT_RT700_MIMXRT798S._wwdt_config.windowValue (C var), 977
drivers_RT_RT700_MIMXRT798S._wwdt_status_flags_t (C enum), 976
drivers_RT_RT700_MIMXRT798S._wwdt_status_flags_t.kWWDT_TimeoutFlag (C enumerator), 976
drivers_RT_RT700_MIMXRT798S._wwdt_status_flags_t.kWWDT_WarningFlag (C enumerator), 976
drivers_RT_RT700_MIMXRT798S._xspi_ahb_access_config (C struct), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahb_access_config.ahbAlignment (C var), 1045
drivers_RT_RT700_MIMXRT798S._xspi_ahb_access_config.ahbErrorPayload (C struct), 1048
drivers_RT_RT700_MIMXRT798S._xspi_ahb_access_config.ahbErrorPayload.highPayload (C var), 1048
drivers_RT_RT700_MIMXRT798S._xspi_ahb_access_config.ahbErrorPayload.lowPayload (C var), 1048
drivers_RT_RT700_MIMXRT798S._xspi_ahb_access_config.ahbSplitSize (C var), 1045
drivers_RT_RT700_MIMXRT798S._xspi_ahb_access_config.ARDSeqIndex (C var), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahb_access_config.buffer (C var), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahb_access_config.enableAHBBufferWriteFlush (C var), 1045

drivers_RT_RT700_MIMXRT798S._xspi_ahb_access_config.enableAHBPrefetch (*C var*), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahb_access_config.enableWriteTerminate (*C var*), 1045
drivers_RT_RT700_MIMXRT798S._xspi_ahb_access_config.ptrAhbWriteConfig (*C var*), 1045
drivers_RT_RT700_MIMXRT798S._xspi_ahb_alignment (*C enum*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_alignment.kXSPI_AhbAlignment1KBLimit (*C enumerator*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_alignment.kXSPI_AhbAlignment256BLimit (*C enumerator*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_alignment.kXSPI_AhbAlignment512BLimit (*C enumerator*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_alignment.kXSPI_AhbAlignmentNoLimit (*C enumerator*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_read_error_info (*C struct*), 1045
drivers_RT_RT700_MIMXRT798S._xspi_ahb_read_error_info.errMasterId (*C var*), 1045
drivers_RT_RT700_MIMXRT798S._xspi_ahb_read_error_info.errorAddr (*C var*), 1045
drivers_RT_RT700_MIMXRT798S._xspi_ahb_request_suspend_info (*C struct*), 1045
drivers_RT_RT700_MIMXRT798S._xspi_ahb_request_suspend_info.address (*C var*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_ahb_request_suspend_info.ahbBufferId (*C var*), 1045
drivers_RT_RT700_MIMXRT798S._xspi_ahb_request_suspend_info.dataLeft (*C var*), 1045
drivers_RT_RT700_MIMXRT798S._xspi_ahb_request_suspend_info.state (*C var*), 1045
drivers_RT_RT700_MIMXRT798S._xspi_ahb_request_suspend_info.subBufferId (*C var*), 1045
drivers_RT_RT700_MIMXRT798S._xspi_ahb_request_suspend_state (*C enum*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_request_suspend_state.kXSPI_AhbReqestResumed (*C enumerator*), 998
drivers_RT_RT700_MIMXRT798S._xspi_ahb_request_suspend_state.kXSPI_AhbRequestNoSuspended (*C enumerator*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_request_suspend_state.kXSPI_AhbRequestNotApplicable (*C enumerator*), 998
drivers_RT_RT700_MIMXRT798S._xspi_ahb_request_suspend_state.kXSPI_AhbRequestSuspended (*C enumerator*), 998
drivers_RT_RT700_MIMXRT798S._xspi_ahb_split_size (*C enum*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_split_size.kXSPI_AhbSplitSize8b (*C enumerator*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_split_size.kXSPI_AhbSplitSize16b (*C enumerator*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_split_size.kXSPI_AhbSplitSize32b (*C enumerator*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_split_size.kXSPI_AhbSplitSize64b (*C enumerator*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_split_size.kXSPI_AhbSplitSizeDisabled (*C enumerator*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ahb_sub_buffer_status (*C enum*), 996
drivers_RT_RT700_MIMXRT798S._xspi_ahb_sub_buffer_status.kXSPI_AhbSubBufferEmpty (*C enumerator*), 996
drivers_RT_RT700_MIMXRT798S._xspi_ahb_sub_buffer_status.kXSPI_AhbSubBufferFull (*C enumerator*), 996
drivers_RT_RT700_MIMXRT798S._xspi_ahb_sub_buffer_status.kXSPI_AhbSubBufferPartiallyFull (*C enumerator*), 996
drivers_RT_RT700_MIMXRT798S._xspi_ahb_write_config (*C struct*), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahb_write_config.ARDSRSeqIndex (*C var*), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahb_write_config.AWRSeqIndex (*C var*), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahb_write_config.blockRead (*C var*), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahb_write_config.blockSequenceWrite (*C var*), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahb_write_config.pageWaitTimeoutValue (*C var*), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahb_write_config.policy (*C var*), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_config (*C struct*), 1043
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_config.enaPri (*C union*), 1048
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_config.enaPri.enableAllMaster (*C var*), 1048

drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_config.enaPri.enablePriority (*C var*), 1048
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_config.masterId (*C var*), 1043
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_config.ptrSubBuffer0Config (*C var*), 1043
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_config.ptrSubBuffer1Config (*C var*), 1043
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_config.ptrSubBuffer2Config (*C var*), 1043
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_config.ptrSubBuffer3Config (*C var*), 1043
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_perf_monitor_result (*C struct*), 1043
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_perf_monitor_result.bufferHitCount (*C var*), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_perf_monitor_result.bufferMissCount (*C var*), 1044
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_sub_buffer_config (*C struct*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_sub_buffer_config.divFactor (*C var*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_sub_buffer_config.enableAhbMonitor (*C var*), 1043
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_sub_buffer_config.endAddr (*C var*), 1043
drivers_RT_RT700_MIMXRT798S._xspi_ahbBuffer_sub_buffer_config.startAddr (*C var*), 1043
drivers_RT_RT700_MIMXRT798S._xspi_byte_order (*C enum*), 998
drivers_RT_RT700_MIMXRT798S._xspi_byte_order.kXSPI_32BitBE (*C enumerator*), 998
drivers_RT_RT700_MIMXRT798S._xspi_byte_order.kXSPI_32BitLE (*C enumerator*), 998
drivers_RT_RT700_MIMXRT798S._xspi_byte_order.kXSPI_64BitBE (*C enumerator*), 998
drivers_RT_RT700_MIMXRT798S._xspi_byte_order.kXSPI_64BitLE (*C enumerator*), 998
drivers_RT_RT700_MIMXRT798S._xspi_cache64_config (*C struct*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_cache64_config.region0Policy (*C var*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_cache64_config.region0TopAddr (*C var*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_cache64_config.region1Policy (*C var*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_cache64_config.region1TopAddr (*C var*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_cache64_config.region2Policy (*C var*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_cache64_policy (*C enum*), 998
drivers_RT_RT700_MIMXRT798S._xspi_cache64_policy.kXSPI_Cache64PolicyNonCacheable (*C enumerator*), 998
drivers_RT_RT700_MIMXRT798S._xspi_cache64_policy.kXSPI_Cache64PolicyWriteBack (*C enumerator*), 998
drivers_RT_RT700_MIMXRT798S._xspi_cache64_policy.kXSPI_Cache64PolicyWriteThrough (*C enumerator*), 998
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag (*C enum*), 987
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagAhbBufferOverflow (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagAhbIllegalBurstSizeError (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagAhbIllegalTransactionError (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagAhbPerformanceMonitorOver (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagAhbReadAddrError (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagAhbTransactionError (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagDataLearningPatternFailure (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagDllAbort (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagDllUnlock (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagIllegalInstrError (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagIpCmdFinished (*C enumerator*), 987

drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagIpCmdTriggerFail (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagPageProgramWait (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagRxBufferOverflow (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagRxBufferWatermarkExceeded (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagTxBufferFill (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_cmd_execution_arbitration_flag.kXSPI_FlagTxBufferUnderRun (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_command_type (*C enum*), 994
drivers_RT_RT700_MIMXRT798S._xspi_command_type.kXSPI_Command (*C enumerator*), 994
drivers_RT_RT700_MIMXRT798S._xspi_command_type.kXSPI_Config (*C enumerator*), 994
drivers_RT_RT700_MIMXRT798S._xspi_command_type.kXSPI_Read (*C enumerator*), 994
drivers_RT_RT700_MIMXRT798S._xspi_command_type.kXSPI_Write (*C enumerator*), 994
drivers_RT_RT700_MIMXRT798S._xspi_config (*C struct*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_config.byteOrder (*C var*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_config.enableDoze (*C var*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_config.ptrAhbAccessConfig (*C var*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_config.ptrIpAccessConfig (*C var*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_data_learning_config (*C struct*), 1036
drivers_RT_RT700_MIMXRT798S._xspi_data_learning_config.deviceSupported (*C var*), 1036
drivers_RT_RT700_MIMXRT798S._xspi_data_learning_config.padSelected (*C var*), 1036
drivers_RT_RT700_MIMXRT798S._xspi_data_learning_config.pattern (*C var*), 1036
drivers_RT_RT700_MIMXRT798S._xspi_data_learning_pad_select (*C enum*), 992
drivers_RT_RT700_MIMXRT798S._xspi_data_learning_pad_select.kXSPI_BothIO1IO3 (*C enumerator*), 993
drivers_RT_RT700_MIMXRT798S._xspi_data_learning_pad_select.kXSPI_OnlyIO1 (*C enumerator*), 993
drivers_RT_RT700_MIMXRT798S._xspi_ddr_data_aligned_clk (*C enum*), 994
drivers_RT_RT700_MIMXRT798S._xspi_ddr_data_aligned_clk.kXSPI_DDRDataAlignedWith2xInternalRefClk (*C enumerator*), 994
drivers_RT_RT700_MIMXRT798S._xspi_ddr_data_aligned_clk.kXSPI_DDRDataAlignedWithInternalRefClk (*C enumerator*), 994
drivers_RT_RT700_MIMXRT798S._xspi_descriptor_lock (*C enum*), 995
drivers_RT_RT700_MIMXRT798S._xspi_descriptor_lock.kXSPI_DescriptorLockDisabled (*C enumerator*), 995
drivers_RT_RT700_MIMXRT798S._xspi_descriptor_lock.kXSPI_DescriptorLockEnabled (*C enumerator*), 995
drivers_RT_RT700_MIMXRT798S._xspi_descriptor_lock.kXSPI_DescriptorLockEnabledExceptMasterId (*C enumerator*), 995
drivers_RT_RT700_MIMXRT798S._xspi_descriptor_lock.kXSPI_DescriptorLockEnabledTillHardReset (*C enumerator*), 995
drivers_RT_RT700_MIMXRT798S._xspi_device_addr_mode (*C enum*), 993
drivers_RT_RT700_MIMXRT798S._xspi_device_addr_mode.kXSPI_Device2ByteAddressable (*C enumerator*), 993
drivers_RT_RT700_MIMXRT798S._xspi_device_addr_mode.kXSPI_Device4ByteAddressable (*C enumerator*), 993
drivers_RT_RT700_MIMXRT798S._xspi_device_addr_mode.kXSPI_DeviceByteAddressable (*C enumerator*), 993
drivers_RT_RT700_MIMXRT798S._xspi_device_config (*C struct*), 1037
drivers_RT_RT700_MIMXRT798S._xspi_device_config.addrMode (*C var*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_device_config.columnAddrWidth (*C var*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_device_config.CSHoldTime (*C var*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_device_config.CSSetupTime (*C var*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_device_config.deviceInterface (*C var*), 1038

drivers_RT_RT700_MIMXRT798S._xspi_device_config.deviceSize (*C var*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_device_config.enableCASInterleaving (*C var*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_device_config.enableCknPad (*C var*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_device_config.interfaceSettings (*C var*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_device_config.ptrDeviceDdrConfig (*C var*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_device_config.ptrDeviceRegInfo (*C var*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_device_config.sampleClkConfig (*C var*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_device_config.xspiRootClk (*C var*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_device_ddr_config (*C struct*), 1037
drivers_RT_RT700_MIMXRT798S._xspi_device_ddr_config.ddrDataAlignedClk (*C var*), 1037
drivers_RT_RT700_MIMXRT798S._xspi_device_ddr_config.enableByteSwapInOctalMode (*C var*), 1037
drivers_RT_RT700_MIMXRT798S._xspi_device_ddr_config.enableDdr (*C var*), 1037
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_settings (*C union*), 1037
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_settings.hyperBusSettings (*C struct*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_settings.hyperBusSettings (*C var*), 1037
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_settings.hyperBusSettings.enableVariableLatency (*C var*), 1048
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_settings.hyperBusSettings.forceBit10To1 (*C var*), 1048
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_settings.hyperBusSettings.pageSize (*C var*), 1048
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_settings.hyperBusSettings.x16Mode (*C var*), 1048
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_settings.strandardExtendedSPISettings (*C struct*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_settings.strandardExtendedSPISettings (*C var*), 1037
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_settings.strandardExtendedSPISettings.pageSize (*C var*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_type (*C enum*), 993
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_type.kXSPI_HyperBus (*C enumerator*), 993
drivers_RT_RT700_MIMXRT798S._xspi_device_interface_type.kXSPI_StrandardExtendedSPI (*C enumerator*), 993
drivers_RT_RT700_MIMXRT798S._xspi_device_status_reg_info (*C struct*), 1035
drivers_RT_RT700_MIMXRT798S._xspi_device_status_reg_info.upperHalfWordSelected (*C var*), 1035
drivers_RT_RT700_MIMXRT798S._xspi_device_status_reg_info.value1Expired (*C var*), 1035
drivers_RT_RT700_MIMXRT798S._xspi_device_status_reg_info.wipLocation (*C var*), 1035
drivers_RT_RT700_MIMXRT798S._xspi_dll_config (*C struct*), 1036
drivers_RT_RT700_MIMXRT798S._xspi_dll_config.dllCustomDelayTapNum (*C var*), 1036
drivers_RT_RT700_MIMXRT798S._xspi_dll_config.dllCustomPara (*C var*), 1036
drivers_RT_RT700_MIMXRT798S._xspi_dll_config.dllMode (*C var*), 1036
drivers_RT_RT700_MIMXRT798S._xspi_dll_config.enableCdl8 (*C var*), 1036
drivers_RT_RT700_MIMXRT798S._xspi_dll_config.useRefValue (*C var*), 1036
drivers_RT_RT700_MIMXRT798S._xspi_dll_mode (*C enum*), 992
drivers_RT_RT700_MIMXRT798S._xspi_dll_mode.kXSPI_AutoUpdateMode (*C enumerator*), 992
drivers_RT_RT700_MIMXRT798S._xspi_dll_mode.kXSPI_BypassMode (*C enumerator*), 992
drivers_RT_RT700_MIMXRT798S._xspi_dll_para (*C union*), 1035
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.autoUpdateModoPara (*C struct*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.autoUpdateModoPara (*C var*), 1036
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.autoUpdateModoPara.enableHighFreq (*C var*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.autoUpdateModoPara.offsetDelayElementCount (*C var*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.autoUpdateModoPara.referenceCounter (*C var*),

1047
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.autoUpdateModoPara.resolution (*C var*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.autoUpdateModoPara.tDiv16OffsetDelayElementCount (*C var*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.bypassModePara (*C struct*), 1046
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.bypassModePara (*C var*), 1035
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.bypassModePara.bypassModeReserved (*C var*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.bypassModePara.delayElementCoarseValue (*C var*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.bypassModePara.delayElementFineValue (*C var*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.bypassModePara.enableHighFreq (*C var*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_dll_para.bypassModePara.offsetDelayElementCount (*C var*), 1047
drivers_RT_RT700_MIMXRT798S._xspi_edma_handle (*C struct*), 1051
drivers_RT_RT700_MIMXRT798S._xspi_edma_ntransfer_size (*C enum*), 1048
drivers_RT_RT700_MIMXRT798S._xspi_edma_ntransfer_size.kXSPI_EDMAnSize1Bytes (*C enumerator*), 1048
drivers_RT_RT700_MIMXRT798S._xspi_edma_ntransfer_size.kXSPI_EDMAnSize2Bytes (*C enumerator*), 1048
drivers_RT_RT700_MIMXRT798S._xspi_edma_ntransfer_size.kXSPI_EDMAnSize4Bytes (*C enumerator*), 1049
drivers_RT_RT700_MIMXRT798S._xspi_edma_ntransfer_size.kXSPI_EDMAnSize8Bytes (*C enumerator*), 1049
drivers_RT_RT700_MIMXRT798S._xspi_edma_ntransfer_size.kXSPI_EDMAnSize32Bytes (*C enumerator*), 1049
drivers_RT_RT700_MIMXRT798S._xspi_error_flag (*C enum*), 985
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ArbitrationWin (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorAllFlags (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorArbitrationLock (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad0Access (*C enumerator*), 986
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad1Access (*C enumerator*), 986
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad2Access (*C enumerator*), 986
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad3Access (*C enumerator*), 986
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad4Access (*C enumerator*), 986
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad5Access (*C enumerator*), 986
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad6Access (*C enumerator*), 986
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad7Access (*C enumerator*), 986
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad8Access (*C enumerator*), 986
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad9Access (*C enumerator*), 986
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad10Access (*C enumerator*), 986
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad11Access (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad12Access (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad13Access (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad14Access (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorFrad15Access (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorIpBusTransfer (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorNoFradMatch (*C enumerator*), 986
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorTg0Ipcr (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorTg0Sfar (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorTimeout (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_error_flag.kXSPI_ErrorWriteLockedReg (*C enumerator*), 987
drivers_RT_RT700_MIMXRT798S._xspi_exclusive_access_lock_mode (*C enum*), 995

drivers_RT_RT700_MIMXRT798S._xspi_exclusive_access_lock_mode.kXSPI_ExclusiveAccessLockDisabled (*C enumerator*), 995
drivers_RT_RT700_MIMXRT798S._xspi_exclusive_access_lock_mode.kXSPI_ExclusiveAccessLockEnabled (*C enumerator*), 995
drivers_RT_RT700_MIMXRT798S._xspi_exclusive_access_lock_mode.kXSPI_ExclusiveAccessLockExceptMasterId (*C enumerator*), 995
drivers_RT_RT700_MIMXRT798S._xspi_external_signal (*C enum*), 992
drivers_RT_RT700_MIMXRT798S._xspi_external_signal.XSPI_SignalIOFA2 (*C enumerator*), 992
drivers_RT_RT700_MIMXRT798S._xspi_external_signal.XSPI_SignalIOFA3 (*C enumerator*), 992
drivers_RT_RT700_MIMXRT798S._xspi_flags (*C enum*), 983
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_AhbAbortErrorFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_AhbBufferOverflowFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_AhbIllegalBurstSizeErrorFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_AhbIllegalTransactionErrorFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_AhbPerformanceMonitorOverflowFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_AhbReadAddressErrorFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_ArbLockTimeoutFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_ArbWinEventFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_DataLearningFailedFlag (*C enumerator*), 983
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_DllAbortFlag (*C enumerator*), 983
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_DllUnlockFlag (*C enumerator*), 983
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad0AccessErrorFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad1AccessErrorFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad2AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad3AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad4AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad5AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad6AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad7AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad8AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad9AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad10AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad11AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad12AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad13AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad14AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_Frad15AccessErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_FradMatchErrorFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_FradnAccErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_IllegalInstructionErrorFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_IpCmdtriggerErrorFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_IpCommandExecutionDoneFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_IpsErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_LockRegErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_PageProgramWaitFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_RxBufferDrainFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_RxBufferOverflowFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_SequenceExecutionTimeoutFlag (*C enumerator*), 984
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_TgnIpcrErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_TgnSfarErrorFlag (*C enumerator*), 985
drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_TxBufferFillFlag (*C enumerator*), 983

drivers_RT_RT700_MIMXRT798S._xspi_flags.kXSPI_TxBufferUnderrunFlag (*C enumerator*), 983
drivers_RT_RT700_MIMXRT798S._xspi_frad_config_t (*C struct*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_frad_config_t.endAddress (*C var*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_frad_config_t.startAddress (*C var*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_frad_tg_config (*C struct*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_frad_tg_config.assignIsValid (*C var*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_frad_tg_config.descriptorLock (*C var*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_frad_tg_config.exclusiveAccessLock (*C var*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_frad_tg_config.tgMasterAccess (*C var*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_frad_transaction_info (*C struct*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_frad_transaction_info.isCompError (*C var*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_frad_transaction_info.isCompValid (*C var*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_frad_transaction_info.isPrivilegeAccess (*C var*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_frad_transaction_info.isSecureAccess (*C var*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_frad_transaction_info.masterId (*C var*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_frad_transaction_info.startAddr (*C var*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_handle (*C struct*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_handle.completionCallback (*C var*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_handle.curTransferSize (*C var*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_handle.data (*C var*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_handle.dataSize (*C var*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_handle.state (*C var*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_handle.tgId (*C var*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_handle.transferTotalSize (*C var*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_handle.userData (*C var*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_hyper_bus_x16_mode (*C enum*), 993
drivers_RT_RT700_MIMXRT798S._xspi_hyper_bus_x16_mode.kXSPI_x16ModeDisable (*C enumerator*), 993
drivers_RT_RT700_MIMXRT798S._xspi_hyper_bus_x16_mode.kXSPI_x16ModeEnabledAddrCmdData (*C enumerator*), 993
drivers_RT_RT700_MIMXRT798S._xspi_hyper_bus_x16_mode.kXSPI_x16ModeEnabledOnlyData (*C enumerator*), 993
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable (*C enum*), 988
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_AhbAbortErrorIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_AhbBufferOverflowIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_AhbIllegalBurstSizeErrorIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_AhbIllegalTransactionErrorIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_AhbPerformanceMonitorOverflowIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_AhbReadAddrErrorIntEnable (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_ArbitrationLockTimeoutErrIntEnable (*C enumerator*), 991
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_ArbitrationWinIntEnable (*C enumerator*), 991
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_DataLearningPatternFailIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_DllUnlockIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad0AccessErrorIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad1AccessErrorIntEnable (*C enumerator*), 990
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad2AccessErrorIntEnable (*C*

enumerator), 990
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad3AccessErrorIntEnable (*C enumerator*), 990
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad4AccessErrorIntEnable (*C enumerator*), 990
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad5AccessErrorIntEnable (*C enumerator*), 990
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad6AccessErrorIntEnable (*C enumerator*), 990
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad7AccessErrorIntEnable (*C enumerator*), 990
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad8AccessErrorIntEnable (*C enumerator*), 990
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad9AccessErrorIntEnable (*C enumerator*), 990
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad10AccessErrorIntEnable (*C enumerator*), 990
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad11AccessErrorIntEnable (*C enumerator*), 991
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad12AccessErrorIntEnable (*C enumerator*), 991
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad13AccessErrorIntEnable (*C enumerator*), 991
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad14AccessErrorIntEnable (*C enumerator*), 991
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Frad15AccessErrorIntEnable (*C enumerator*), 991
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_IllegalInstrErrorIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_IpCmdTrigFailErrorIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_IpsErrIntEnable (*C enumerator*), 991
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_LockRegWriteErrIntEnable (*C enumerator*), 991
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_NoFradMatchErrorIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_PageProgramWaitIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_RxBufferDrainIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_RxBufferOverflowIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Tg0IpcrErrIntEnable (*C enumerator*), 991
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_Tg0SfarErrIntEnable (*C enumerator*), 991
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_TimeoutErrIntEnable (*C enumerator*), 991
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_TransactionFinishIntEnable (*C enumerator*), 988
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_TxBufferFillIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_interrupt_enable.kXSPI_TxBufferUnderrunIntEnable (*C enumerator*), 989
drivers_RT_RT700_MIMXRT798S._xspi_ip_access_config (*C struct*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_ip_access_config.ipAccessTimeoutValue (*C var*), 1041
drivers_RT_RT700_MIMXRT798S._xspi_ip_access_config.ptrSfpFradConfig (*C var*), 1041

drivers_RT_RT700_MIMXRT798S._xspi_ip_access_config.ptrSfpMdadConfig (*C var*), 1041
drivers_RT_RT700_MIMXRT798S._xspi_ip_access_config.sfpArbitrationLockTimeoutValue (*C var*), 1041
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr (*C enum*), 981
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_CADDR_DDR (*C enumerator*), 983
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_CADDR_SDR (*C enumerator*), 983
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_DDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_DUMMY_SDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_JMP_ON_CS (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_JUMP_TO_SEQ (*C enumerator*), 983
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_LEARN_DDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_MODE2_DDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_MODE2_SDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_MODE4_DDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_MODE4_SDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_MODE_DDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_MODE_SDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_RADDR_DDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_RADDR_SDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_READ_DDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_READ_SDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_SDR (*C enumerator*), 981
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_STOP (*C enumerator*), 981
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_WRITE_DDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_lut_instr.kXSPI_Command_WRITE_SDR (*C enumerator*), 982
drivers_RT_RT700_MIMXRT798S._xspi_mdad_config (*C struct*), 1038
drivers_RT_RT700_MIMXRT798S._xspi_mdad_config.assignIsValid (*C var*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_mdad_config.enableDescriptorLock (*C var*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_mdad_config.mask (*C var*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_mdad_config.maskType (*C var*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_mdad_config.masterIdReference (*C var*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_mdad_config.secureAttribute (*C var*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_mdad_error_reason (*C enum*), 994
drivers_RT_RT700_MIMXRT798S._xspi_mdad_error_reason.kXSPI_ErrorBothDataSizeSeqId (*C enumerator*), 995
drivers_RT_RT700_MIMXRT798S._xspi_mdad_error_reason.kXSPI_ErrorOnlyDataSize (*C enumerator*), 994
drivers_RT_RT700_MIMXRT798S._xspi_mdad_error_reason.kXSPI_ErrorOnlySeqId (*C enumerator*), 994
drivers_RT_RT700_MIMXRT798S._xspi_mdad_mask_type (*C enum*), 994

drivers_RT_RT700_MIMXRT798S._xspi_mdad_mask_type.kXSPI_MdadMaskTypeAnd (*C enumerator*), 994
drivers_RT_RT700_MIMXRT798S._xspi_mdad_mask_type.kXSPI_MdadMaskTypeOr (*C enumerator*), 994
drivers_RT_RT700_MIMXRT798S._xspi_otfad_prefetch_boundary (*C enum*), 998
drivers_RT_RT700_MIMXRT798S._xspi_otfad_prefetch_boundary.kXSPI_OTFADPrefetchBoundary1K (*C enumerator*), 998
drivers_RT_RT700_MIMXRT798S._xspi_otfad_prefetch_boundary.kXSPI_OTFADPrefetchNoBoundary (*C enumerator*), 998
drivers_RT_RT700_MIMXRT798S._xspi_pad (*C enum*), 983
drivers_RT_RT700_MIMXRT798S._xspi_pad.kXSPI_1PAD (*C enumerator*), 983
drivers_RT_RT700_MIMXRT798S._xspi_pad.kXSPI_2PAD (*C enumerator*), 983
drivers_RT_RT700_MIMXRT798S._xspi_pad.kXSPI_4PAD (*C enumerator*), 983
drivers_RT_RT700_MIMXRT798S._xspi_pad.kXSPI_8PAD (*C enumerator*), 983
drivers_RT_RT700_MIMXRT798S._xspi_ppw_flag_clear_policy (*C enum*), 996
drivers_RT_RT700_MIMXRT798S._xspi_ppw_flag_clear_policy.kXSPI_HardwareClearPPWFlag (*C enumerator*), 997
drivers_RT_RT700_MIMXRT798S._xspi_ppw_flag_clear_policy.kXSPI_SoftwareClearPPWFlag (*C enumerator*), 996
drivers_RT_RT700_MIMXRT798S._xspi_sample_clk_config (*C struct*), 1036
drivers_RT_RT700_MIMXRT798S._xspi_sample_clk_config.dllConfig (*C var*), 1037
drivers_RT_RT700_MIMXRT798S._xspi_sample_clk_config.enableDQSLatency (*C var*), 1037
drivers_RT_RT700_MIMXRT798S._xspi_sample_clk_config.sampleClkSource (*C var*), 1037
drivers_RT_RT700_MIMXRT798S._xspi_sample_clk_source (*C enum*), 992
drivers_RT_RT700_MIMXRT798S._xspi_sample_clk_source.kXSPI_SampleClkFromDqsPadLoopback (*C enumerator*), 992
drivers_RT_RT700_MIMXRT798S._xspi_sample_clk_source.kXSPI_SampleClkFromExternalDQS (*C enumerator*), 992
drivers_RT_RT700_MIMXRT798S._xspi_sample_clk_source.kXSPI_SampleClkFromHalfSpeedDummyPadLoopback (*C enumerator*), 992
drivers_RT_RT700_MIMXRT798S._xspi_sample_clk_source.kXSPI_SampleClkFromInvertedFullySpeedDummyPad (*C enumerator*), 992
drivers_RT_RT700_MIMXRT798S._xspi_sample_clk_source.kXSPI_SampleClkFromNonInvertedFullySpeedDummy (*C enumerator*), 992
drivers_RT_RT700_MIMXRT798S._xspi_secure_attribute (*C enum*), 995
drivers_RT_RT700_MIMXRT798S._xspi_secure_attribute.kXSPI_AttributeMasterNonsecureOnly (*C enumerator*), 995
drivers_RT_RT700_MIMXRT798S._xspi_secure_attribute.kXSPI_AttributeMasterNonsecureSecureBoth (*C enumerator*), 995
drivers_RT_RT700_MIMXRT798S._xspi_secure_attribute.kXSPI_AttributeMasterSecureOnly (*C enumerator*), 995
drivers_RT_RT700_MIMXRT798S._xspi_sfp_frad_config (*C struct*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_sfp_frad_config.fradConfig (*C var*), 1040
drivers_RT_RT700_MIMXRT798S._xspi_sfp_mdad_config (*C struct*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_sfp_mdad_config.tgMdad (*C var*), 1039
drivers_RT_RT700_MIMXRT798S._xspi_subbuffer_division_factor (*C enum*), 996
drivers_RT_RT700_MIMXRT798S._xspi_subbuffer_division_factor.kXSPI_Division0 (*C enumerator*), 996
drivers_RT_RT700_MIMXRT798S._xspi_subbuffer_division_factor.kXSPI_Division2 (*C enumerator*), 996
drivers_RT_RT700_MIMXRT798S._xspi_subbuffer_division_factor.kXSPI_Division4 (*C enumerator*), 996
drivers_RT_RT700_MIMXRT798S._xspi_subbuffer_division_factor.kXSPI_Division8 (*C enumerator*), 996
drivers_RT_RT700_MIMXRT798S._xspi_subbuffer_division_factor.kXSPI_Division16 (*C enumerator*), 996
drivers_RT_RT700_MIMXRT798S._xspi_target_group (*C enum*), 996
drivers_RT_RT700_MIMXRT798S._xspi_target_group.kXSPI_TargetGroup0 (*C enumerator*), 996

drivers_RT_RT700_MIMXRT798S._xspi_target_group.kXSPI_TargetGroup1 (*C enumerator*), 996
drivers_RT_RT700_MIMXRT798S._xspi_tg_add_write_status (*C struct*), 1041
drivers_RT_RT700_MIMXRT798S._xspi_tg_add_write_status.managerId (*C var*), 1041
drivers_RT_RT700_MIMXRT798S._xspi_tg_add_write_status.privilegedWrite (*C var*), 1041
drivers_RT_RT700_MIMXRT798S._xspi_tg_add_write_status.secureWrite (*C var*), 1041
drivers_RT_RT700_MIMXRT798S._xspi_transfer (*C struct*), 1041
drivers_RT_RT700_MIMXRT798S._xspi_transfer.cmdType (*C var*), 1041
drivers_RT_RT700_MIMXRT798S._xspi_transfer.data (*C var*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_transfer.dataSize (*C var*), 1042
drivers_RT_RT700_MIMXRT798S._xspi_transfer.deviceAddress (*C var*), 1041
drivers_RT_RT700_MIMXRT798S._xspi_transfer.seqIndex (*C var*), 1041
drivers_RT_RT700_MIMXRT798S._xspi_transfer.targetGroup (*C var*), 1042
drivers_RT_RT700_MIMXRT798S.acmp_channel_config_t (*C type*), 200
drivers_RT_RT700_MIMXRT798S.ACMP_ClearStatusFlags (*C function*), 197
drivers_RT_RT700_MIMXRT798S.acmp_config_t (*C type*), 200
drivers_RT_RT700_MIMXRT798S.acmp_dac_config_t (*C type*), 200
drivers_RT_RT700_MIMXRT798S.acmp_dac_work_mode_t (*C type*), 200
drivers_RT_RT700_MIMXRT798S.ACMP_Deinit (*C function*), 195
drivers_RT_RT700_MIMXRT798S.ACMP_DisableInterrupts (*C function*), 197
drivers_RT_RT700_MIMXRT798S.acmp_discrete_mode_config_t (*C type*), 200
drivers_RT_RT700_MIMXRT798S.ACMP_Enable (*C function*), 195
drivers_RT_RT700_MIMXRT798S.ACMP_EnableDMA (*C function*), 196
drivers_RT_RT700_MIMXRT798S.ACMP_EnableInterrupts (*C function*), 197
drivers_RT_RT700_MIMXRT798S.ACMP_EnableLinkToDAC (*C function*), 195
drivers_RT_RT700_MIMXRT798S.acmp_filter_config_t (*C type*), 200
drivers_RT_RT700_MIMXRT798S.ACMP_GetDefaultConfig (*C function*), 195
drivers_RT_RT700_MIMXRT798S.ACMP_GetDefaultDiscreteModeConfig (*C function*), 197
drivers_RT_RT700_MIMXRT798S.ACMP_GetStatusFlags (*C function*), 197
drivers_RT_RT700_MIMXRT798S.acmp_hysteresis_mode_t (*C type*), 200
drivers_RT_RT700_MIMXRT798S.ACMP_Init (*C function*), 195
drivers_RT_RT700_MIMXRT798S.acmp_offset_mode_t (*C type*), 199
drivers_RT_RT700_MIMXRT798S.acmp_port_input_t (*C type*), 200
drivers_RT_RT700_MIMXRT798S.acmp_reference_voltage_source_t (*C type*), 200
drivers_RT_RT700_MIMXRT798S.ACMP_SetChannelConfig (*C function*), 196
drivers_RT_RT700_MIMXRT798S.ACMP_SetDACConfig (*C function*), 196
drivers_RT_RT700_MIMXRT798S.ACMP_SetDiscreteModeConfig (*C function*), 197
drivers_RT_RT700_MIMXRT798S.ACMP_SetFilterConfig (*C function*), 196
drivers_RT_RT700_MIMXRT798S.ADC_OFSTRIM_OFSTRIM_MAX (*C macro*), 592
drivers_RT_RT700_MIMXRT798S.ADC_OFSTRIM_OFSTRIM_SIGN (*C macro*), 592
drivers_RT_RT700_MIMXRT798S.ADC_RSTS (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.ancillaryData (*C var*), 791
drivers_RT_RT700_MIMXRT798S.ancillaryLength (*C var*), 791
drivers_RT_RT700_MIMXRT798S.argc (*C var*), 329
drivers_RT_RT700_MIMXRT798S.argv (*C var*), 330
drivers_RT_RT700_MIMXRT798S.arm2ezhv_intctl_t (*C type*), 327
drivers_RT_RT700_MIMXRT798S.ARRAY_SIZE (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.AT_CACHE_LINE_SECTION (*C macro*), 544
drivers_RT_RT700_MIMXRT798S.AT_CACHE_LINE_SECTION_INIT (*C macro*), 544
drivers_RT_RT700_MIMXRT798S.AT_NONCACHEABLE_SECTION (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.AT_NONCACHEABLE_SECTION_ALIGN (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.AT_NONCACHEABLE_SECTION_ALIGN_INIT (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.AT_NONCACHEABLE_SECTION_INIT (*C macro*), 542, 543
drivers_RT_RT700_MIMXRT798S.AT_QUICKACCESS_SECTION_CODE (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.AT_QUICKACCESS_SECTION_DATA (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.AT_QUICKACCESS_SECTION_DATA_ALIGN (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.audio_pll_mult (*C var*), 267
drivers_RT_RT700_MIMXRT798S.audio_pll_mult_t (*C type*), 246

drivers_RT_RT700_MIMXRT798S.audio_pll_src (*C var*), 267
drivers_RT_RT700_MIMXRT798S.audio_pll_src_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.AUDIOPLL0CLKSEL_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.AUDIOPLL0LOCKTIMEDIV2_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.AUDIOVDD1CLKSEL_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.autoStart (*C var*), 537
drivers_RT_RT700_MIMXRT798S.B (*C var*), 840, 842
drivers_RT_RT700_MIMXRT798S.boot_interface (*C var*), 842
drivers_RT_RT700_MIMXRT798S.bootloader_copyright (*C function*), 841
drivers_RT_RT700_MIMXRT798S.bootloader_user_entry (*C function*), 841
drivers_RT_RT700_MIMXRT798S.bootloader_version (*C function*), 841
drivers_RT_RT700_MIMXRT798S.bypass (*C var*), 267
drivers_RT_RT700_MIMXRT798S.CACHE64_CleanCache (*C function*), 203
drivers_RT_RT700_MIMXRT798S.CACHE64_CleanCacheByRange (*C function*), 203
drivers_RT_RT700_MIMXRT798S.CACHE64_CleanInvalidateCache (*C function*), 204
drivers_RT_RT700_MIMXRT798S.CACHE64_CleanInvalidateCacheByRange (*C function*), 204
drivers_RT_RT700_MIMXRT798S.CACHE64_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.cache64_config_t (*C type*), 205
drivers_RT_RT700_MIMXRT798S.CACHE64_DisableCache (*C function*), 203
drivers_RT_RT700_MIMXRT798S.CACHE64_EnableCache (*C function*), 203
drivers_RT_RT700_MIMXRT798S.CACHE64_GetDefaultConfig (*C function*), 202
drivers_RT_RT700_MIMXRT798S.CACHE64_GetInstance (*C function*), 202
drivers_RT_RT700_MIMXRT798S.CACHE64_GetInstanceByAddr (*C function*), 202
drivers_RT_RT700_MIMXRT798S.CACHE64_Init (*C function*), 202
drivers_RT_RT700_MIMXRT798S.CACHE64_InvalidateCache (*C function*), 203
drivers_RT_RT700_MIMXRT798S.CACHE64_InvalidateCacheByRange (*C function*), 203
drivers_RT_RT700_MIMXRT798S.CACHE64_LINESIZE_BYTE (*C macro*), 205
drivers_RT_RT700_MIMXRT798S.cache64_policy_t (*C type*), 205
drivers_RT_RT700_MIMXRT798S.CACHE64_REGION_ALIGNMENT (*C macro*), 206
drivers_RT_RT700_MIMXRT798S.CACHE64_REGION_NUM (*C macro*), 206
drivers_RT_RT700_MIMXRT798S.CACHE64POLSEL_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.CACHE_LINE_DATA (*C macro*), 544
drivers_RT_RT700_MIMXRT798S.cap (*C var*), 267
drivers_RT_RT700_MIMXRT798S.CDOG_Add (*C function*), 207
drivers_RT_RT700_MIMXRT798S.CDOG_Add1 (*C function*), 207
drivers_RT_RT700_MIMXRT798S.CDOG_Add16 (*C function*), 208
drivers_RT_RT700_MIMXRT798S.CDOG_Add256 (*C function*), 208
drivers_RT_RT700_MIMXRT798S.CDOG_Check (*C function*), 207
drivers_RT_RT700_MIMXRT798S.CDOG_CLOCKS (*C macro*), 258
drivers_RT_RT700_MIMXRT798S.cdog_config_t (*C struct*), 210
drivers_RT_RT700_MIMXRT798S.CDOG_Deinit (*C function*), 206
drivers_RT_RT700_MIMXRT798S.CDOG_GetDefaultConfig (*C function*), 206
drivers_RT_RT700_MIMXRT798S.CDOG_Init (*C function*), 206
drivers_RT_RT700_MIMXRT798S.CDOG_ReadPersistent (*C function*), 208
drivers_RT_RT700_MIMXRT798S.CDOG_Set (*C function*), 207
drivers_RT_RT700_MIMXRT798S.CDOG_Start (*C function*), 207
drivers_RT_RT700_MIMXRT798S.CDOG_Stop (*C function*), 206
drivers_RT_RT700_MIMXRT798S.CDOG_Sub (*C function*), 208
drivers_RT_RT700_MIMXRT798S.CDOG_Sub1 (*C function*), 208
drivers_RT_RT700_MIMXRT798S.CDOG_Sub16 (*C function*), 208
drivers_RT_RT700_MIMXRT798S.CDOG_Sub256 (*C function*), 208
drivers_RT_RT700_MIMXRT798S.CDOG_WritePersistent (*C function*), 208
drivers_RT_RT700_MIMXRT798S.chunk_v3_t (*C type*), 880
drivers_RT_RT700_MIMXRT798S.CKGATING (*C var*), 821
drivers_RT_RT700_MIMXRT798S.clearStreamBuf (*C var*), 537
drivers_RT_RT700_MIMXRT798S.CLK_CTL0_PSCCTL0 (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_CTL0_PSCCTL1 (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_CTL0_PSCCTL2 (*C macro*), 261

drivers_RT_RT700_MIMXRT798S.CLK_CTL0_PSCCTL3 (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_CTL0_PSCCTL4 (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_CTL0_PSCCTL5 (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_CTL1_PSCCTL0 (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_CTL1_PSCCTL1 (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_CTL2_PSCCTL0 (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_CTL3_PSCCTL0 (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_CTL4_PSCCTL0 (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_CTL4_PSCCTL1 (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_GATE_ABSTRACT_BITS_SHIFT (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.CLK_GATE_ABSTRACT_REG_OFFSET (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.CLK_GATE_BIT_SHIFT_MASK (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.CLK_GATE_BIT_SHIFT_SHIFT (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.CLK_GATE_DEFINE (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.CLK_GATE_REG_OFFSET_MASK (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.CLK_GATE_REG_OFFSET_SHIFT (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.CLK_MUX_CHOICE_OFFSET_SHIFT (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_MUX_DISABLE_OUTPUT_MASK (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLK_MUX_INST_INDEX_SHIFT (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLKCTL0_INDEX (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.CLKCTL0_ONE_SRC_CLKSLICE_ENABLE (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLKCTL0_TUPLE_MUXA (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.CLKCTL0_TUPLE_MUXA_NONE (*C macro*), 266
drivers_RT_RT700_MIMXRT798S.CLKCTL1_INDEX (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.CLKCTL1_TUPLE_MUXA (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.CLKCTL1_TUPLE_MUXA_NONE (*C macro*), 266
drivers_RT_RT700_MIMXRT798S.CLKCTL2_INDEX (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.CLKCTL2_TUPLE_MUXA (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.CLKCTL2_TUPLE_MUXA_NONE (*C macro*), 266
drivers_RT_RT700_MIMXRT798S.CLKCTL3_INDEX (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.CLKCTL3_ONE_SRC_CLKSLICE_ENABLE (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLKCTL3_TUPLE_MUXA (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.CLKCTL3_TUPLE_MUXA_NONE (*C macro*), 266
drivers_RT_RT700_MIMXRT798S.CLKCTL4_INDEX (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.CLKCTL4_ONE_SRC_CLKSLICE_ENABLE (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.CLKCTL4_TUPLE_MUXA (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.CLKCTL4_TUPLE_MUXA_NONE (*C macro*), 266
drivers_RT_RT700_MIMXRT798S.CLKCTL_TUPLE_REG (*C macro*), 266
drivers_RT_RT700_MIMXRT798S.CLKCTL_TUPLE_SEL (*C macro*), 266
drivers_RT_RT700_MIMXRT798S.clock_attach_id_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.CLOCK_AttachClk (*C function*), 247
drivers_RT_RT700_MIMXRT798S.clock_audio_pll_config_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.CLOCK_ClearFroFlags (*C function*), 250
drivers_RT_RT700_MIMXRT798S.CLOCK_ConfigFroTrim (*C function*), 248
drivers_RT_RT700_MIMXRT798S.CLOCK_DeinitAudioPfd (*C function*), 254
drivers_RT_RT700_MIMXRT798S.CLOCK_DeinitAudioPll (*C function*), 253
drivers_RT_RT700_MIMXRT798S.CLOCK_DeinitMainPfd (*C function*), 253
drivers_RT_RT700_MIMXRT798S.CLOCK_DeinitMainPll (*C function*), 253
drivers_RT_RT700_MIMXRT798S.CLOCK_DeinitSysPfd (*C macro*), 258
drivers_RT_RT700_MIMXRT798S.CLOCK_DisableClock (*C function*), 247
drivers_RT_RT700_MIMXRT798S.CLOCK_DisableFro (*C function*), 248
drivers_RT_RT700_MIMXRT798S.CLOCK_DisableOsc32K (*C function*), 254
drivers_RT_RT700_MIMXRT798S.CLOCK_DisableUsbhs0PhyPllClock (*C function*), 258
drivers_RT_RT700_MIMXRT798S.clock_div_name_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.clock_domain_enable_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableAudioPllPfdClkForDomain (*C function*), 252
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableAudioPllVcoClkForDomain (*C function*), 252
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableClock (*C function*), 247

drivers_RT_RT700_MIMXRT798S.CLOCK_EnableFro2ClkForDomain (*C function*), 252
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableFroAutoTuning (*C function*), 248
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableFroClkFreq (*C function*), 249
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableFroClkFreqCloseLoop (*C function*), 249
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableFroClkOutput (*C function*), 248
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableFroFastStartup (*C function*), 250
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableMainPllPfdClkForDomain (*C function*), 252
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableOsc32K (*C function*), 254
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableSysOscClk (*C function*), 251
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableUsbhs0Clock (*C function*), 257
drivers_RT_RT700_MIMXRT798S.CLOCK_EnableUsbhs0PhyPllClock (*C function*), 257
drivers_RT_RT700_MIMXRT798S.clock_fro_config_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.clock_fro_output_en_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.CLOCK_FroFineTune (*C function*), 250
drivers_RT_RT700_MIMXRT798S.CLOCK_FroTuneOnce (*C function*), 249
drivers_RT_RT700_MIMXRT798S.CLOCK_GetAcmpClkFreq (*C function*), 255
drivers_RT_RT700_MIMXRT798S.CLOCK_GetAdcClkFreq (*C function*), 257
drivers_RT_RT700_MIMXRT798S.CLOCK_GetAudioPfdFreq (*C function*), 251
drivers_RT_RT700_MIMXRT798S.CLOCK_GetAudioPllFreq (*C function*), 251
drivers_RT_RT700_MIMXRT798S.CLOCK_GetCoreSysClkFreq (*C function*), 255
drivers_RT_RT700_MIMXRT798S.CLOCK_GetCTimerClkFreq (*C function*), 256
drivers_RT_RT700_MIMXRT798S.CLOCK_GetFlexioClkFreq (*C function*), 256
drivers_RT_RT700_MIMXRT798S.CLOCK_GetFreq (*C function*), 248
drivers_RT_RT700_MIMXRT798S.CLOCK_GetFroClkFreq (*C function*), 250
drivers_RT_RT700_MIMXRT798S.CLOCK_GetFroFlags (*C function*), 250
drivers_RT_RT700_MIMXRT798S.CLOCK_GetHifi1ClkFreq (*C function*), 248
drivers_RT_RT700_MIMXRT798S.CLOCK_GetI3cClkFreq (*C function*), 255
drivers_RT_RT700_MIMXRT798S.CLOCK_GetLcdifClkFreq (*C function*), 257
drivers_RT_RT700_MIMXRT798S.CLOCK_GetLPFlexCommClkFreq (*C function*), 256
drivers_RT_RT700_MIMXRT798S.CLOCK_GetLPI2cClkFreq (*C function*), 256
drivers_RT_RT700_MIMXRT798S.CLOCK_GetLpOscFreq (*C function*), 254
drivers_RT_RT700_MIMXRT798S.CLOCK_GetLPSpiClkFreq (*C function*), 256
drivers_RT_RT700_MIMXRT798S.CLOCK_GetMainPfdFreq (*C function*), 251
drivers_RT_RT700_MIMXRT798S.CLOCK_GetMainPllFreq (*C function*), 250
drivers_RT_RT700_MIMXRT798S.CLOCK_GetMclkInClkFreq (*C function*), 254
drivers_RT_RT700_MIMXRT798S.CLOCK_GetMediaMainClkFreq (*C function*), 252
drivers_RT_RT700_MIMXRT798S.CLOCK_GetMediaVddnClkFreq (*C function*), 252
drivers_RT_RT700_MIMXRT798S.CLOCK_GetMicfilClkFreq (*C function*), 256
drivers_RT_RT700_MIMXRT798S.CLOCK_GetMipiDphyClkFreq (*C function*), 257
drivers_RT_RT700_MIMXRT798S.CLOCK_GetMipiDphyEscRxClkFreq (*C function*), 257
drivers_RT_RT700_MIMXRT798S.CLOCK_GetMipiDphyEscTxClkFreq (*C function*), 257
drivers_RT_RT700_MIMXRT798S.CLOCK_GetOsc32KFreq (*C function*), 254
drivers_RT_RT700_MIMXRT798S.CLOCK_GetOSTimerClkFreq (*C function*), 256
drivers_RT_RT700_MIMXRT798S.CLOCK_GetSaiClkFreq (*C function*), 255
drivers_RT_RT700_MIMXRT798S.CLOCK_GetSdAdcClkFreq (*C function*), 257
drivers_RT_RT700_MIMXRT798S.CLOCK_GetSenseAudioClkFreq (*C function*), 248
drivers_RT_RT700_MIMXRT798S.CLOCK_GetSenseBaseClkFreq (*C function*), 251
drivers_RT_RT700_MIMXRT798S.CLOCK_GetSenseMainClkFreq (*C function*), 250
drivers_RT_RT700_MIMXRT798S.CLOCK_GetSysOscFreq (*C function*), 254
drivers_RT_RT700_MIMXRT798S.CLOCK_GetSystickClkFreq (*C function*), 255
drivers_RT_RT700_MIMXRT798S.CLOCK_GetUsbClkFreq (*C function*), 255
drivers_RT_RT700_MIMXRT798S.CLOCK_GetUsdhcClkFreq (*C function*), 257
drivers_RT_RT700_MIMXRT798S.CLOCK_GetUtickClkFreq (*C function*), 255
drivers_RT_RT700_MIMXRT798S.CLOCK_GetVdd1ClockOutClkFreq (*C function*), 256
drivers_RT_RT700_MIMXRT798S.CLOCK_GetVdd2MediaBaseClkFreq (*C function*), 252
drivers_RT_RT700_MIMXRT798S.CLOCK_GetVddnComBaseClkFreq (*C function*), 251
drivers_RT_RT700_MIMXRT798S.CLOCK_GetVddnMediaBaseClkFreq (*C function*), 251
drivers_RT_RT700_MIMXRT798S.CLOCK_GetVgpuClkFreq (*C function*), 257

drivers_RT_RT700_MIMXRT798S.CLOCK_GetWakeClk32KFreq (*C function*), 254
drivers_RT_RT700_MIMXRT798S.CLOCK_GetWdtClkFreq (*C function*), 255
drivers_RT_RT700_MIMXRT798S.CLOCK_GetXspiClkFreq (*C function*), 255
drivers_RT_RT700_MIMXRT798S.CLOCK_GetXtalInClkFreq (*C function*), 251
drivers_RT_RT700_MIMXRT798S.CLOCK_InitAudioPfd (*C function*), 253
drivers_RT_RT700_MIMXRT798S.CLOCK_InitAudioPll (*C function*), 253
drivers_RT_RT700_MIMXRT798S.CLOCK_InitMainPfd (*C function*), 253
drivers_RT_RT700_MIMXRT798S.CLOCK_InitMainPll (*C function*), 253
drivers_RT_RT700_MIMXRT798S.CLOCK_InitSysPfd (*C macro*), 258
drivers_RT_RT700_MIMXRT798S.clock_ip_name_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.clock_main_pll_config_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.clock_name_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.clock_osc32k_config_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.clock_pfd_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.CLOCK_Set32kClkinFreq (*C function*), 254
drivers_RT_RT700_MIMXRT798S.CLOCK_SetClkDiv (*C function*), 248
drivers_RT_RT700_MIMXRT798S.CLOCK_SetClkinFreq (*C function*), 254
drivers_RT_RT700_MIMXRT798S.CLOCK_SetMclkFreq (*C function*), 255
drivers_RT_RT700_MIMXRT798S.CLOCK_SetXtalFreq (*C function*), 254
drivers_RT_RT700_MIMXRT798S.clock_usb_phy_src_t (*C type*), 245
drivers_RT_RT700_MIMXRT798S.clock_usb_src_t (*C type*), 245
drivers_RT_RT700_MIMXRT798S.CMP_C0_CFx_MASK (*C macro*), 200
drivers_RT_RT700_MIMXRT798S.CMP_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.coarseTrimEn (*C var*), 266
drivers_RT_RT700_MIMXRT798S.colorType (*C var*), 790
drivers_RT_RT700_MIMXRT798S.COMMONVDDNCLKDIV_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.COMMONVDDNCLKSEL_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.COMNBASECLKSEL_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.completionCallback (*C var*), 1051
drivers_RT_RT700_MIMXRT798S.config (*C var*), 537
drivers_RT_RT700_MIMXRT798S.config_memory (*C struct*), 882
drivers_RT_RT700_MIMXRT798S.config_memory.address (*C var*), 882
drivers_RT_RT700_MIMXRT798S.copy (*C struct*), 882
drivers_RT_RT700_MIMXRT798S.copy_memory_expansion (*C struct*), 882
drivers_RT_RT700_MIMXRT798S.core (*C var*), 536
drivers_RT_RT700_MIMXRT798S.count (*C var*), 1051
drivers_RT_RT700_MIMXRT798S.COUNT_TO_MSEC (*C macro*), 540, 541
drivers_RT_RT700_MIMXRT798S.COUNT_TO_USEC (*C macro*), 540, 541
drivers_RT_RT700_MIMXRT798S.crc_bits_t (*C type*), 268
drivers_RT_RT700_MIMXRT798S.CRC_CLOCKS (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.crc_config_t (*C type*), 269
drivers_RT_RT700_MIMXRT798S.CRC_Deinit (*C function*), 269
drivers_RT_RT700_MIMXRT798S.CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT (*C macro*), 270
drivers_RT_RT700_MIMXRT798S.CRC_Get16bitResult (*C function*), 270
drivers_RT_RT700_MIMXRT798S.CRC_Get32bitResult (*C function*), 269
drivers_RT_RT700_MIMXRT798S.CRC_GetDefaultConfig (*C function*), 269
drivers_RT_RT700_MIMXRT798S.CRC_Init (*C function*), 269
drivers_RT_RT700_MIMXRT798S.crc_result_t (*C type*), 268
drivers_RT_RT700_MIMXRT798S.CRC_RSTS (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.CRC_WriteData (*C function*), 269
drivers_RT_RT700_MIMXRT798S.CTIMER5CLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.CTIMER5FCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.CTIMER6CLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.CTIMER6FCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.CTIMER7CLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.CTIMER7FCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.ctimer_callback_t (*C type*), 277

drivers_RT_RT700_MIMXRT798S.ctimer_callback_type_t (*C enum*), 276
drivers_RT_RT700_MIMXRT798S.ctimer_callback_type_t.kCTIMER_MultipleCallback (*C enumerator*), 276
drivers_RT_RT700_MIMXRT798S.ctimer_callback_type_t.kCTIMER_SingleCallback (*C enumerator*), 276
drivers_RT_RT700_MIMXRT798S.ctimer_capture_channel_t (*C type*), 276
drivers_RT_RT700_MIMXRT798S.ctimer_capture_edge_t (*C type*), 276
drivers_RT_RT700_MIMXRT798S.CTIMER_ClearStatusFlags (*C function*), 273
drivers_RT_RT700_MIMXRT798S.CTIMER_CLOCKS (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.ctimer_config_t (*C type*), 277
drivers_RT_RT700_MIMXRT798S.CTIMER_Deinit (*C function*), 271
drivers_RT_RT700_MIMXRT798S.CTIMER_DisableInterrupts (*C function*), 273
drivers_RT_RT700_MIMXRT798S.CTIMER_EnableFallingEdgeCapture (*C function*), 280
drivers_RT_RT700_MIMXRT798S.CTIMER_EnableInterrupts (*C function*), 273
drivers_RT_RT700_MIMXRT798S.CTIMER_EnableMatchChannelReload (*C function*), 279
drivers_RT_RT700_MIMXRT798S.CTIMER_EnableResetMatchChannel (*C function*), 279
drivers_RT_RT700_MIMXRT798S.CTIMER_EnableRisingEdgeCapture (*C function*), 279
drivers_RT_RT700_MIMXRT798S.CTIMER_EnableStopMatchChannel (*C function*), 279
drivers_RT_RT700_MIMXRT798S.ctimer_external_match_t (*C type*), 277
drivers_RT_RT700_MIMXRT798S.CTIMER_GetCaptureValue (*C function*), 279
drivers_RT_RT700_MIMXRT798S.CTIMER_GetDefaultConfig (*C function*), 271
drivers_RT_RT700_MIMXRT798S.CTIMER_GetEnabledInterrupts (*C function*), 273
drivers_RT_RT700_MIMXRT798S.CTIMER_GetOutputMatchStatus (*C function*), 277
drivers_RT_RT700_MIMXRT798S.CTIMER_GetStatusFlags (*C function*), 273
drivers_RT_RT700_MIMXRT798S.CTIMER_GetTimerCountValue (*C function*), 278
drivers_RT_RT700_MIMXRT798S.CTIMER_Init (*C function*), 271
drivers_RT_RT700_MIMXRT798S.ctimer_interrupt_enable_t (*C type*), 277
drivers_RT_RT700_MIMXRT798S.ctimer_match_config_t (*C type*), 277
drivers_RT_RT700_MIMXRT798S.ctimer_match_output_control_t (*C type*), 277
drivers_RT_RT700_MIMXRT798S.ctimer_match_t (*C type*), 277
drivers_RT_RT700_MIMXRT798S.CTIMER_RegisterCallBack (*C function*), 278
drivers_RT_RT700_MIMXRT798S.CTIMER_Reset (*C function*), 278
drivers_RT_RT700_MIMXRT798S.CTIMER_RSTS (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.CTIMER_SetPrescale (*C function*), 279
drivers_RT_RT700_MIMXRT798S.CTIMER_SetShadowValue (*C function*), 280
drivers_RT_RT700_MIMXRT798S.CTIMER_SetupCapture (*C function*), 278
drivers_RT_RT700_MIMXRT798S.CTIMER_SetupMatch (*C function*), 277
drivers_RT_RT700_MIMXRT798S.CTIMER_SetupPwm (*C function*), 272
drivers_RT_RT700_MIMXRT798S.CTIMER_SetupPwmPeriod (*C function*), 271
drivers_RT_RT700_MIMXRT798S.CTIMER_StartTimer (*C function*), 273
drivers_RT_RT700_MIMXRT798S.ctimer_status_flags_t (*C type*), 277
drivers_RT_RT700_MIMXRT798S.CTIMER_StopTimer (*C function*), 273
drivers_RT_RT700_MIMXRT798S.ctimer_timer_mode_t (*C type*), 277
drivers_RT_RT700_MIMXRT798S.CTIMER_UpdatePwmDutycycle (*C function*), 272
drivers_RT_RT700_MIMXRT798S.CTIMER_UpdatePwmPulsePeriod (*C function*), 272
drivers_RT_RT700_MIMXRT798S.current_mode (*C var*), 840
drivers_RT_RT700_MIMXRT798S.dataEndianness (*C var*), 821
drivers_RT_RT700_MIMXRT798S.DCACHE_CleanByRange (*C function*), 204, 979
drivers_RT_RT700_MIMXRT798S.DCACHE_CleanInvalidateByRange (*C function*), 205, 979
drivers_RT_RT700_MIMXRT798S.DCACHE_InvalidateByRange (*C function*), 204, 979
drivers_RT_RT700_MIMXRT798S.DCDC (*C var*), 817
drivers_RT_RT700_MIMXRT798S.DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM (*C macro*), 538
drivers_RT_RT700_MIMXRT798S.DEBUG_CONSOLE_DEVICE_TYPE_IUART (*C macro*), 538
drivers_RT_RT700_MIMXRT798S.DEBUG_CONSOLE_DEVICE_TYPE_LPSCI (*C macro*), 538
drivers_RT_RT700_MIMXRT798S.DEBUG_CONSOLE_DEVICE_TYPE_LPUART (*C macro*), 538

drivers_RT_RT700_MIMXRT798S.DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.DEBUG_CONSOLE_DEVICE_TYPE_NONE (*C macro*), 538
drivers_RT_RT700_MIMXRT798S.DEBUG_CONSOLE_DEVICE_TYPE_QSCI (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.DEBUG_CONSOLE_DEVICE_TYPE_SWO (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.DEBUG_CONSOLE_DEVICE_TYPE_UART (*C macro*), 538
drivers_RT_RT700_MIMXRT798S.DEBUG_CONSOLE_DEVICE_TYPE_USBCDC (*C macro*), 538
drivers_RT_RT700_MIMXRT798S.DEBUG_CONSOLE_DEVICE_TYPE_VUSART (*C macro*), 538
drivers_RT_RT700_MIMXRT798S.denominator (*C var*), 266, 267
drivers_RT_RT700_MIMXRT798S.destAddr (*C var*), 282, 329
drivers_RT_RT700_MIMXRT798S.DisableDeepSleepIRQ (*C function*), 809
drivers_RT_RT700_MIMXRT798S.DisableGlobalIRQ (*C function*), 554
drivers_RT_RT700_MIMXRT798S.DisableIRQ (*C function*), 554
drivers_RT_RT700_MIMXRT798S.DisableMachineModeInt (*C function*), 554
drivers_RT_RT700_MIMXRT798S.DMA_ATTR_DMOD (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_ATTR_DMOD_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_ATTR_DSIZE (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_ATTR_DSIZE_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_ATTR_DSIZE_SHIFT (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_ATTR_SMOD (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_ATTR_SMOD_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_ATTR_SSIZE (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_ATTR_SSIZE_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_ATTR_SSIZE_SHIFT (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_BITER_ELINKNO_BITER_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_BITER_ELINKYES_ELINK_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_BITER_ELINKYES_LINKCH (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_BITER_ELINKYES_LINKCH_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CH_CSR_SIGNEXT_MASK (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CH_CSR_SIGNEXT_SHIFT (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CH_CSR_SWAP_MASK (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CH_CSR_SWAP_SHIFT (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CH_MATTR_RCACHE (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CH_MATTR_WCACHE (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CH_MUX_SOURCE (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CH_SBR_EMI_MASK (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CH_SBR_EMI_SHIFT (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CH_SBR_INSTR_MASK (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CH_SBR_INSTR_SHIFT (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CITER_ELINKNO_CITER_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CITER_ELINKNO_CITER_SHIFT (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CITER_ELINKNO_ELINK_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CITER_ELINKYES_CITER_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CITER_ELINKYES_CITER_SHIFT (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CITER_ELINKYES_ELINK_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CITER_ELINKYES_LINKCH (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CITER_ELINKYES_LINKCH_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CLEAR_DONE_STATUS (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_CLEAR_ERROR_STATUS (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_CLEAR_INT_STATUS (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_CORE_MP_CSR_EDBG (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CORE_MP_CSR_EDBG_MASK (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CORE_MP_CSR_ERCA (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CORE_MP_CSR_ERCA_MASK (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CORE_MP_CSR_GCLC (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CORE_MP_CSR_GCLC_MASK (*C macro*), 317

drivers_RT_RT700_MIMXRT798S.DMA_CORE_MP_CSR_GMRC (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CORE_MP_CSR_GMRC_MASK (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CORE_MP_CSR_HAE (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CORE_MP_CSR_HAE_MASK (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CORE_MP_CSR_HALT (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CORE_MP_CSR_HALT_MASK (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CSR_BWC (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CSR_BWC_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CSR_DREQ (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CSR_DREQ_MASK (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CSR_ESG_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CSR_INTHALF_MASK (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CSR_INTMAJOR_MASK (*C macro*), 317
drivers_RT_RT700_MIMXRT798S.DMA_CSR_MAJORELINK_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CSR_MAJORLINKCH (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_CSR_MAJORLINKCH_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_CSR_START_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_DCHPRI_INDEX (*C macro*), 311
drivers_RT_RT700_MIMXRT798S.DMA_DISABLE_ERROR_INT (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_DISABLE_HALF_INT (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_DISABLE_MAJOR_INT (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ENABLE_ERROR_INT (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ENABLE_HALF_INT (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ENABLE_MAJOR_INT (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ERR_DAE_FLAG (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ERR_DBE_FLAG (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_ERR_DOE_FLAG (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ERR_ECX_FLAG (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ERR_ERRCHAN_FLAG (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ERR_FLAG (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ERR_NCE_FLAG (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ERR_SAE_FLAG (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ERR_SBE_FLAG (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ERR_SGE_FLAG (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_ERR_SOE_FLAG (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_GET_DONE_STATUS (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_GET_ERROR_STATUS (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_GET_INT_STATUS (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.DMA_NBYTES_MLOFFNO_NBYTES (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_NBYTES_MLOFFNO_NBYTES_MASK (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_NBYTES_MLOFFYES_DMLOE (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_NBYTES_MLOFFYES_DMLOE_MASK (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_NBYTES_MLOFFYES_MLOFF (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_NBYTES_MLOFFYES_MLOFF_MASK (*C macro*), 318
drivers_RT_RT700_MIMXRT798S.DMA_NBYTES_MLOFFYES_NBYTES (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_NBYTES_MLOFFYES_SMLOE (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DMA_NBYTES_MLOFFYES_SMLOE_MASK (*C macro*), 319
drivers_RT_RT700_MIMXRT798S.DPHYCLKDIV_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.DPHYCLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.DPHYESCCLKSEL_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.DPHYESCRXCLKDIV_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.DPHYESCTXCLKDIV_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.dsi_callback_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.dsi_config_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.dsi_dbi_pixel_format_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.DSI_Deinit (*C function*), 700
drivers_RT_RT700_MIMXRT798S.DSI_DeinitDphy (*C function*), 701
drivers_RT_RT700_MIMXRT798S.DSI_DisableInterrupts (*C function*), 702

drivers_RT_RT700_MIMXRT798S.dsi_dphy_config_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.DSI_DPHY_PLL_VCO_MAX (*C macro*), 928
drivers_RT_RT700_MIMXRT798S.DSI_DPHY_PLL_VCO_MIN (*C macro*), 928
drivers_RT_RT700_MIMXRT798S.dsi_dpi_blp_mode_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.dsi_dpi_color_coding_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.dsi_dpi_config_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.dsi_dpi_pixel_packet_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.dsi_dpi_video_mode_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.DSI_EnableInterrupts (*C function*), 701
drivers_RT_RT700_MIMXRT798S.DSI_GetAndClearHostStatus (*C function*), 704
drivers_RT_RT700_MIMXRT798S.DSI_GetAndClearInterruptStatus (*C function*), 702
drivers_RT_RT700_MIMXRT798S.DSI_GetApbStatus (*C function*), 704
drivers_RT_RT700_MIMXRT798S.DSI_GetDefaultConfig (*C function*), 700
drivers_RT_RT700_MIMXRT798S.DSI_GetDphyDefaultConfig (*C function*), 701
drivers_RT_RT700_MIMXRT798S.DSI_GetEccRxErrorPosition (*C function*), 704
drivers_RT_RT700_MIMXRT798S.DSI_GetRxErrorStatus (*C function*), 704
drivers_RT_RT700_MIMXRT798S.DSI_GetRxPacketHeader (*C function*), 704
drivers_RT_RT700_MIMXRT798S.DSI_GetRxPacketType (*C function*), 705
drivers_RT_RT700_MIMXRT798S.DSI_GetRxPacketVirtualChannel (*C function*), 705
drivers_RT_RT700_MIMXRT798S.DSI_GetRxPacketWordCount (*C function*), 705
drivers_RT_RT700_MIMXRT798S.DSI_GetUlpsStatus (*C function*), 701
drivers_RT_RT700_MIMXRT798S.dsi_handle_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.DSI_Init (*C function*), 699
drivers_RT_RT700_MIMXRT798S.DSI_InitDphy (*C function*), 700
drivers_RT_RT700_MIMXRT798S.DSI_ReadApbRxPayload (*C function*), 703
drivers_RT_RT700_MIMXRT798S.dsi_rx_data_type_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.DSI_SendApbPacket (*C function*), 703
drivers_RT_RT700_MIMXRT798S.DSI_SetApbPacketControl (*C function*), 702
drivers_RT_RT700_MIMXRT798S.DSI_SetDbiPixelFifoSendLevel (*C function*), 702
drivers_RT_RT700_MIMXRT798S.DSI_SetDbiPixelFormat (*C function*), 702
drivers_RT_RT700_MIMXRT798S.DSI_SetDbiPixelPayloadSize (*C function*), 702
drivers_RT_RT700_MIMXRT798S.DSI_SetDpiConfig (*C function*), 700
drivers_RT_RT700_MIMXRT798S.DSI_SetUlpsStatus (*C function*), 701
drivers_RT_RT700_MIMXRT798S.dsi_transfer_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.DSI_TransferAbort (*C function*), 706
drivers_RT_RT700_MIMXRT798S.DSI_TransferBlocking (*C function*), 705
drivers_RT_RT700_MIMXRT798S.DSI_TransferCreateHandle (*C function*), 706
drivers_RT_RT700_MIMXRT798S.DSI_TransferHandleIRQ (*C function*), 706
drivers_RT_RT700_MIMXRT798S.DSI_TransferNonBlocking (*C function*), 706
drivers_RT_RT700_MIMXRT798S.dsi_tx_data_type_t (*C type*), 717
drivers_RT_RT700_MIMXRT798S.DSI_WriteApbTxPayload (*C function*), 703
drivers_RT_RT700_MIMXRT798S.DSI_WriteApbTxPayloadExt (*C function*), 703
drivers_RT_RT700_MIMXRT798S.dsp_copy_image_t (*C type*), 281
drivers_RT_RT700_MIMXRT798S.DSP_CopyImage (*C function*), 282
drivers_RT_RT700_MIMXRT798S.DSP_Deinit (*C function*), 281
drivers_RT_RT700_MIMXRT798S.DSP_Init (*C function*), 281
drivers_RT_RT700_MIMXRT798S.DSP_INT_PMUX_ID (*C macro*), 512
drivers_RT_RT700_MIMXRT798S.DSP_SetVecRemap (*C function*), 282
drivers_RT_RT700_MIMXRT798S.DSP_Start (*C function*), 282
drivers_RT_RT700_MIMXRT798S.dsp_static_vec_sel_t (*C type*), 281
drivers_RT_RT700_MIMXRT798S.DSP_Stop (*C function*), 282
drivers_RT_RT700_MIMXRT798S.edma4_core_tcd_t (*C type*), 317
drivers_RT_RT700_MIMXRT798S.edma5_core_tcd_t (*C type*), 317
drivers_RT_RT700_MIMXRT798S.EDMA_AbortTransfer (*C function*), 300
drivers_RT_RT700_MIMXRT798S.EDMA_ALLOCATE_TCD (*C macro*), 311
drivers_RT_RT700_MIMXRT798S.edma_bandwidth_t (*C type*), 309
drivers_RT_RT700_MIMXRT798S.edma_callback (*C type*), 310
drivers_RT_RT700_MIMXRT798S.edma_channel_access_type_t (*C type*), 309

drivers_RT_RT700_MIMXRT798S.EDMA_CHANNEL_ARRAY_STEP (*C macro*), 326
drivers_RT_RT700_MIMXRT798S.EDMA_CHANNEL_BASE (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.edma_channel_config_t (*C type*), 309
drivers_RT_RT700_MIMXRT798S.edma_channel_link_type_t (*C type*), 309
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute (*C enum*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute_t (*C type*), 309
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelNoWriteNoReadCacheableBu
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelNoWriteNoReadCacheableNo
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelNoWriteNoReadNoCacheBuff
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelNoWriteNoReadNoCacheNoB
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelNoWriteReadCacheableBuffer
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelNoWriteReadCacheableNoBu
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelNoWriteReadNoCacheBuffera
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelNoWriteReadNoCacheNoBuff
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelWriteNoReadCacheableBuffer
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelWriteNoReadCacheableNoBu
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelWriteNoReadNoCacheBuffera
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelWriteNoReadNoCacheNoBuff
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelWriteReadCacheableBufferab
(*C enumerator*), 308
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelWriteReadCacheableNoBuffer
(*C enumerator*), 308
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelWriteReadNoCacheBufferable
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.edma_channel_memory_attribute.kEDMA_ChannelWriteReadNoCacheNoBuffer
(*C enumerator*), 307
drivers_RT_RT700_MIMXRT798S.EDMA_CHANNEL_OFFSET (*C macro*), 326
drivers_RT_RT700_MIMXRT798S.edma_channel_Preemption_config_t (*C type*), 309
drivers_RT_RT700_MIMXRT798S.edma_channel_protection_level_t (*C type*), 309
drivers_RT_RT700_MIMXRT798S.edma_channel_swap_size_t (*C type*), 309
drivers_RT_RT700_MIMXRT798S.EDMA_ChannelType (*C type*), 317
drivers_RT_RT700_MIMXRT798S.EDMA_ClearChannelStatusFlags (*C function*), 296
drivers_RT_RT700_MIMXRT798S.edma_config_t (*C type*), 310
drivers_RT_RT700_MIMXRT798S.EDMA_ConfigChannelSoftwareTCD (*C function*), 288
drivers_RT_RT700_MIMXRT798S.EDMA_ConfigChannelSoftwareTCDExt (*C function*), 291
drivers_RT_RT700_MIMXRT798S.EDMA_CORE_BASE (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.edma_core_channel_t (*C type*), 316
drivers_RT_RT700_MIMXRT798S.edma_core_mp_t (*C type*), 316
drivers_RT_RT700_MIMXRT798S.edma_core_tcd_t (*C type*), 317
drivers_RT_RT700_MIMXRT798S.EDMA_CreateHandle (*C function*), 296
drivers_RT_RT700_MIMXRT798S.EDMA_Deinit (*C function*), 283
drivers_RT_RT700_MIMXRT798S.EDMA_DisableChannelInterrupts (*C function*), 288
drivers_RT_RT700_MIMXRT798S.EDMA_DisableChannelRequest (*C function*), 295
drivers_RT_RT700_MIMXRT798S.EDMA_DisableRequest (*C function*), 325
drivers_RT_RT700_MIMXRT798S.EDMA_EnableAsyncRequest (*C function*), 287
drivers_RT_RT700_MIMXRT798S.EDMA_EnableAutoStopRequest (*C function*), 287

drivers_RT_RT700_MIMXRT798S.EDMA_EnableChannelInterrupts (*C function*), 287
drivers_RT_RT700_MIMXRT798S.EDMA_EnableChannelMasterIDReplication (*C function*), 285
drivers_RT_RT700_MIMXRT798S.EDMA_EnableChannelRequest (*C function*), 295
drivers_RT_RT700_MIMXRT798S.EDMA_EnableRequest (*C function*), 325
drivers_RT_RT700_MIMXRT798S.EDMA_GetChannelStatusFlags (*C function*), 296
drivers_RT_RT700_MIMXRT798S.EDMA_GetChannelSystemBusInformation (*C function*), 284
drivers_RT_RT700_MIMXRT798S.EDMA_GetDefaultConfig (*C function*), 283
drivers_RT_RT700_MIMXRT798S.EDMA_GetErrorStatusFlags (*C function*), 295
drivers_RT_RT700_MIMXRT798S.EDMA_GetNextTCDAddress (*C function*), 301
drivers_RT_RT700_MIMXRT798S.EDMA_GetRemainingMajorLoopCount (*C function*), 295
drivers_RT_RT700_MIMXRT798S.EDMA_GetUnusedTCDNumber (*C function*), 300
drivers_RT_RT700_MIMXRT798S.edma_handle_t (*C type*), 311
drivers_RT_RT700_MIMXRT798S.EDMA_HandleIRQ (*C function*), 301
drivers_RT_RT700_MIMXRT798S.EDMA_Init (*C function*), 282
drivers_RT_RT700_MIMXRT798S.EDMA_InitChannel (*C function*), 283
drivers_RT_RT700_MIMXRT798S.EDMA_InstallTCD (*C function*), 283
drivers_RT_RT700_MIMXRT798S.EDMA_InstallTCDMemory (*C function*), 296
drivers_RT_RT700_MIMXRT798S.edma_minor_offset_config_t (*C type*), 309
drivers_RT_RT700_MIMXRT798S.edma_modulo_t (*C type*), 309
drivers_RT_RT700_MIMXRT798S.EDMA_MP_BASE (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.EDMA_PrepareTransfer (*C function*), 297
drivers_RT_RT700_MIMXRT798S.EDMA_PrepareTransferConfig (*C function*), 297
drivers_RT_RT700_MIMXRT798S.EDMA_PrepareTransferTCD (*C function*), 298
drivers_RT_RT700_MIMXRT798S.EDMA_ResetChannel (*C function*), 285
drivers_RT_RT700_MIMXRT798S.EDMA_SetBandWidth (*C function*), 287
drivers_RT_RT700_MIMXRT798S.EDMA_SetCallback (*C function*), 297
drivers_RT_RT700_MIMXRT798S.EDMA_SetChannelAccessType (*C function*), 284
drivers_RT_RT700_MIMXRT798S.EDMA_SetChannelLink (*C function*), 286
drivers_RT_RT700_MIMXRT798S.EDMA_SetChannelMemoryAttribute (*C function*), 283
drivers_RT_RT700_MIMXRT798S.EDMA_SetChannelMux (*C function*), 284
drivers_RT_RT700_MIMXRT798S.EDMA_SetChannelPreemptionConfig (*C function*), 286
drivers_RT_RT700_MIMXRT798S.EDMA_SetChannelProtectionLevel (*C function*), 285
drivers_RT_RT700_MIMXRT798S.EDMA_SetChannelSignExtension (*C function*), 284
drivers_RT_RT700_MIMXRT798S.EDMA_SetChannelSwapSize (*C function*), 284
drivers_RT_RT700_MIMXRT798S.EDMA_SetMajorOffsetConfig (*C function*), 288
drivers_RT_RT700_MIMXRT798S.EDMA_SetMinorOffsetConfig (*C function*), 286
drivers_RT_RT700_MIMXRT798S.EDMA_SetModulo (*C function*), 287
drivers_RT_RT700_MIMXRT798S.EDMA_SetTransferConfig (*C function*), 285
drivers_RT_RT700_MIMXRT798S.EDMA_StartTransfer (*C function*), 300
drivers_RT_RT700_MIMXRT798S.EDMA_StopTransfer (*C function*), 300
drivers_RT_RT700_MIMXRT798S.EDMA_SubmitLoopTransfer (*C function*), 299
drivers_RT_RT700_MIMXRT798S.EDMA_SubmitTransfer (*C function*), 299
drivers_RT_RT700_MIMXRT798S.EDMA_SubmitTransferTCD (*C function*), 298
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_ALIGN_SIZE (*C macro*), 320
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_ATTR (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_BASE (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_BITER (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_CITER (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_CSR (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_DADDR (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_DLAST_SGA (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_DOFF (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_NBYTES (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_SADDR (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_SLAST (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_SOFF (*C macro*), 321
drivers_RT_RT700_MIMXRT798S.edma_tcd_t (*C type*), 310
drivers_RT_RT700_MIMXRT798S.EDMA_TCD_TYPE (*C macro*), 321

drivers_RT_RT700_MIMXRT798S.edma_tcd_type_t (*C type*), 317
drivers_RT_RT700_MIMXRT798S.EDMA_TcdDisableInterrupts (*C function*), 291
drivers_RT_RT700_MIMXRT798S.EDMA_TcdDisableInterruptsExt (*C function*), 294
drivers_RT_RT700_MIMXRT798S.EDMA_TcdEnableAutoStopRequest (*C function*), 291
drivers_RT_RT700_MIMXRT798S.EDMA_TcdEnableAutoStopRequestExt (*C function*), 294
drivers_RT_RT700_MIMXRT798S.EDMA_TcdEnableInterrupts (*C function*), 291
drivers_RT_RT700_MIMXRT798S.EDMA_TcdEnableInterruptsExt (*C function*), 294
drivers_RT_RT700_MIMXRT798S.EDMA_TcdInit (*C function*), 301
drivers_RT_RT700_MIMXRT798S.EDMA_TcdReset (*C function*), 288
drivers_RT_RT700_MIMXRT798S.EDMA_TcdResetExt (*C function*), 292
drivers_RT_RT700_MIMXRT798S.EDMA_TcdSetBandWidth (*C function*), 290
drivers_RT_RT700_MIMXRT798S.EDMA_TcdSetBandWidthExt (*C function*), 293
drivers_RT_RT700_MIMXRT798S.EDMA_TcdSetChannelLink (*C function*), 290
drivers_RT_RT700_MIMXRT798S.EDMA_TcdSetChannelLinkExt (*C function*), 293
drivers_RT_RT700_MIMXRT798S.EDMA_TcdSetMajorOffsetConfig (*C function*), 291
drivers_RT_RT700_MIMXRT798S.EDMA_TcdSetMajorOffsetConfigExt (*C function*), 294
drivers_RT_RT700_MIMXRT798S.EDMA_TcdSetMinorOffsetConfig (*C function*), 289
drivers_RT_RT700_MIMXRT798S.EDMA_TcdSetMinorOffsetConfigExt (*C function*), 293
drivers_RT_RT700_MIMXRT798S.EDMA_TcdSetModulo (*C function*), 290
drivers_RT_RT700_MIMXRT798S.EDMA_TcdSetModuloExt (*C function*), 294
drivers_RT_RT700_MIMXRT798S.EDMA_TcdSetTransferConfig (*C function*), 289
drivers_RT_RT700_MIMXRT798S.EDMA_TcdSetTransferConfigExt (*C function*), 292
drivers_RT_RT700_MIMXRT798S.EDMA_TCDType (*C type*), 317
drivers_RT_RT700_MIMXRT798S.edma_transfer_config_t (*C type*), 310
drivers_RT_RT700_MIMXRT798S.edma_transfer_size_t (*C type*), 309
drivers_RT_RT700_MIMXRT798S.edma_transfer_type_t (*C type*), 309
drivers_RT_RT700_MIMXRT798S.EDMA_TriggerChannelStart (*C function*), 295
drivers_RT_RT700_MIMXRT798S.EDMA_Type (*C type*), 317
drivers_RT_RT700_MIMXRT798S.enable (*C var*), 537, 790
drivers_RT_RT700_MIMXRT798S.enableAncillary (*C var*), 790
drivers_RT_RT700_MIMXRT798S.EnableDeepSleepIRQ (*C function*), 808
drivers_RT_RT700_MIMXRT798S.EnableGlobalIRQ (*C function*), 554
drivers_RT_RT700_MIMXRT798S.enableInt (*C var*), 266
drivers_RT_RT700_MIMXRT798S.EnableIRQ (*C function*), 554
drivers_RT_RT700_MIMXRT798S.EnableIRQWithPriority (*C function*), 555
drivers_RT_RT700_MIMXRT798S.enableLp (*C var*), 537
drivers_RT_RT700_MIMXRT798S.EnableMachineModeInt (*C function*), 554
drivers_RT_RT700_MIMXRT798S.enableVcoOut (*C var*), 267
drivers_RT_RT700_MIMXRT798S.endianMode (*C var*), 537
drivers_RT_RT700_MIMXRT798S.EUSBCLKSRC24MCLKSEL_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.EUSBFCLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.EVENT_COUNT_STABLE_TIMEOUT (*C macro*), 931
drivers_RT_RT700_MIMXRT798S.ezhv2arm_int_chan_t (*C type*), 327
drivers_RT_RT700_MIMXRT798S.EZHV_API_MAX_ARG (*C macro*), 329
drivers_RT_RT700_MIMXRT798S.ezhv_api_t (*C type*), 327
drivers_RT_RT700_MIMXRT798S.EZHV_Boot (*C function*), 328
drivers_RT_RT700_MIMXRT798S.ezhv_callback_t (*C type*), 327
drivers_RT_RT700_MIMXRT798S.EZHV_ClearEzhv2ArmIntChan (*C function*), 329
drivers_RT_RT700_MIMXRT798S.ezhv_copy_image_t (*C type*), 327
drivers_RT_RT700_MIMXRT798S.EZHV_Deinit (*C function*), 328
drivers_RT_RT700_MIMXRT798S.EZHV_DisableArm2EzhvInt (*C function*), 329
drivers_RT_RT700_MIMXRT798S.EZHV_DisableEzhv2ArmIntChan (*C function*), 329
drivers_RT_RT700_MIMXRT798S.EZHV_DriverIRQHandler (*C function*), 329
drivers_RT_RT700_MIMXRT798S.EZHV_EnableArm2EzhvInt (*C function*), 328
drivers_RT_RT700_MIMXRT798S.EZHV_EnableEzhv2ArmIntChan (*C function*), 329
drivers_RT_RT700_MIMXRT798S.EZHV_GetEzhv2ArmIntChan (*C function*), 329
drivers_RT_RT700_MIMXRT798S.EZHV_GetEzhvHaltStatusFlag (*C function*), 329
drivers_RT_RT700_MIMXRT798S.EZHV_GetEzhvStopStatusFlag (*C function*), 329

drivers_RT_RT700_MIMXRT798S.EZHV_GetEzhvWaitStatusFlag (*C function*), 329
drivers_RT_RT700_MIMXRT798S.EZHV_GetEzhvWakeupStatusFlag (*C function*), 329
drivers_RT_RT700_MIMXRT798S.EZHV_GetParaAddr (*C function*), 328
drivers_RT_RT700_MIMXRT798S.EZHV_Init (*C function*), 327
drivers_RT_RT700_MIMXRT798S.EZHV_InitWithoutFirmware (*C function*), 328
drivers_RT_RT700_MIMXRT798S.EZHV_InstallFirmware (*C function*), 328
drivers_RT_RT700_MIMXRT798S.EZHV_INT_CHAN_NUM (*C macro*), 330
drivers_RT_RT700_MIMXRT798S.EZHV_M_EXT_INT (*C macro*), 330
drivers_RT_RT700_MIMXRT798S.ezhv_param_t (*C type*), 327
drivers_RT_RT700_MIMXRT798S.EZHV_SetCallback (*C function*), 328
drivers_RT_RT700_MIMXRT798S.EZHV_SetPara (*C function*), 328
drivers_RT_RT700_MIMXRT798S.EZHV_SHARED_DATA_ADDR (*C macro*), 330
drivers_RT_RT700_MIMXRT798S.EZHV_WakeUpEzhv (*C function*), 328
drivers_RT_RT700_MIMXRT798S.ezhvApi (*C var*), 329
drivers_RT_RT700_MIMXRT798S.fastStartupEn (*C var*), 266
drivers_RT_RT700_MIMXRT798S.FC17FCLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.FC17FCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.FC18FCLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.FC18FCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.FC19FCLKDIV_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.FC19FCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.FC20FCLKDIV_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.FC20FCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.fill_memory (*C struct*), 882
drivers_RT_RT700_MIMXRT798S.fill_memory_expansion (*C struct*), 882
drivers_RT_RT700_MIMXRT798S.fill_memory_expansion.pattern (*C var*), 882
drivers_RT_RT700_MIMXRT798S.FLASH_run_context_t (*C type*), 837
drivers_RT_RT700_MIMXRT798S.FLEXIO_CAMERA_ClearStatusFlags (*C function*), 331
drivers_RT_RT700_MIMXRT798S.flexio_camera_config_t (*C type*), 332
drivers_RT_RT700_MIMXRT798S.FLEXIO_CAMERA_Deinit (*C function*), 330
drivers_RT_RT700_MIMXRT798S.FLEXIO_CAMERA_DisableInterrupt (*C function*), 331
drivers_RT_RT700_MIMXRT798S.FLEXIO_CAMERA_Enable (*C function*), 331
drivers_RT_RT700_MIMXRT798S.FLEXIO_CAMERA_EnableInterrupt (*C function*), 331
drivers_RT_RT700_MIMXRT798S.FLEXIO_CAMERA_EnableRxDMA (*C function*), 331
drivers_RT_RT700_MIMXRT798S.FLEXIO_CAMERA_GetDefaultConfig (*C function*), 330
drivers_RT_RT700_MIMXRT798S.FLEXIO_CAMERA_GetRxBufferAddress (*C function*), 332
drivers_RT_RT700_MIMXRT798S.FLEXIO_CAMERA_GetStatusFlags (*C function*), 331
drivers_RT_RT700_MIMXRT798S.FLEXIO_CAMERA_Init (*C function*), 330
drivers_RT_RT700_MIMXRT798S.FLEXIO_CAMERA_PARALLEL_DATA_WIDTH (*C macro*), 333
drivers_RT_RT700_MIMXRT798S.flexio_camera_transfer_t (*C type*), 332
drivers_RT_RT700_MIMXRT798S.FLEXIO_CAMERA_Type (*C type*), 332
drivers_RT_RT700_MIMXRT798S.FLEXIO_ClearPortOutput (*C function*), 340
drivers_RT_RT700_MIMXRT798S.FLEXIO_ClearPortStatus (*C function*), 342
drivers_RT_RT700_MIMXRT798S.FLEXIO_ClearShifterErrorFlags (*C function*), 338
drivers_RT_RT700_MIMXRT798S.FLEXIO_ClearShifterStatusFlags (*C function*), 338
drivers_RT_RT700_MIMXRT798S.FLEXIO_ClearTimerStatusFlags (*C function*), 339
drivers_RT_RT700_MIMXRT798S.FLEXIO_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.flexio_config_t (*C type*), 350
drivers_RT_RT700_MIMXRT798S.FLEXIO_ConfigPinOverride (*C function*), 341
drivers_RT_RT700_MIMXRT798S.FLEXIO_Deinit (*C function*), 334
drivers_RT_RT700_MIMXRT798S.FLEXIO_DisableShifterErrorInterrupts (*C function*), 337
drivers_RT_RT700_MIMXRT798S.FLEXIO_DisableShifterStatusInterrupts (*C function*), 337
drivers_RT_RT700_MIMXRT798S.FLEXIO_DisableTimerStatusInterrupts (*C function*), 338
drivers_RT_RT700_MIMXRT798S.FLEXIO_Enable (*C function*), 335
drivers_RT_RT700_MIMXRT798S.FLEXIO_EnablePinOutput (*C function*), 341
drivers_RT_RT700_MIMXRT798S.FLEXIO_EnableShifterErrorInterrupts (*C function*), 337
drivers_RT_RT700_MIMXRT798S.FLEXIO_EnableShifterStatusDMA (*C function*), 339

drivers_RT_RT700_MIMXRT798S.FLEXIO_EnableShifterStatusInterrupts (*C function*), 336
drivers_RT_RT700_MIMXRT798S.FLEXIO_EnableTimerStatusInterrupts (*C function*), 337
drivers_RT_RT700_MIMXRT798S.FLEXIO_GetDefaultConfig (*C function*), 334
drivers_RT_RT700_MIMXRT798S.FLEXIO_GetInstance (*C function*), 335
drivers_RT_RT700_MIMXRT798S.FLEXIO_GetPinOverride (*C function*), 341
drivers_RT_RT700_MIMXRT798S.FLEXIO_GetPinStatus (*C function*), 341
drivers_RT_RT700_MIMXRT798S.FLEXIO_GetShifterBufferAddress (*C function*), 339
drivers_RT_RT700_MIMXRT798S.FLEXIO_GetShifterErrorFlags (*C function*), 338
drivers_RT_RT700_MIMXRT798S.FLEXIO_GetShifterState (*C function*), 335
drivers_RT_RT700_MIMXRT798S.FLEXIO_GetShifterStatusFlags (*C function*), 338
drivers_RT_RT700_MIMXRT798S.FLEXIO_GetTimerStatusFlags (*C function*), 339
drivers_RT_RT700_MIMXRT798S.flexio_gpio_config_t (*C type*), 350
drivers_RT_RT700_MIMXRT798S.flexio_gpio_direction_t (*C type*), 350
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_CheckForBusyBus (*C function*), 367
drivers_RT_RT700_MIMXRT798S.flexio_i2c_direction_t (*C type*), 374
drivers_RT_RT700_MIMXRT798S.flexio_i2c_master_config_t (*C type*), 374
drivers_RT_RT700_MIMXRT798S.flexio_i2c_master_handle_t (*C type*), 374
drivers_RT_RT700_MIMXRT798S.flexio_i2c_master_transfer_callback_t (*C type*), 374
drivers_RT_RT700_MIMXRT798S.flexio_i2c_master_transfer_t (*C type*), 374
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterAbortStop (*C function*), 369
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterClearStatusFlags (*C function*), 368
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterDeinit (*C function*), 367
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterDisableInterrupts (*C function*), 368
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterEnable (*C function*), 368
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterEnableAck (*C function*), 369
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterEnableInterrupts (*C function*), 368
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterGetDefaultConfig (*C function*), 368
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterGetStatusFlags (*C function*), 368
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterInit (*C function*), 367
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterReadBlocking (*C function*), 371
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterReadByte (*C function*), 370
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterRepeatedStart (*C function*), 369
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterSetBaudRate (*C function*), 369
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterSetTransferCount (*C function*), 370
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterStart (*C function*), 369
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterStop (*C function*), 369
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterTransferAbort (*C function*), 372
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterTransferBlocking (*C function*), 371
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterTransferCreateHandle (*C function*), 371
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterTransferGetCount (*C function*), 372
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterTransferHandleIRQ (*C function*), 373
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterTransferNonBlocking (*C function*), 372
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterWriteBlocking (*C function*), 370
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_MasterWriteByte (*C function*), 370
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2C_Type (*C type*), 374
drivers_RT_RT700_MIMXRT798S.flexio_i2s_callback_t (*C type*), 385
drivers_RT_RT700_MIMXRT798S.flexio_i2s_config_t (*C type*), 385
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_Deinit (*C function*), 377
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_DisableInterrupts (*C function*), 378
drivers_RT_RT700_MIMXRT798S.flexio_i2s_edma_callback_t (*C type*), 357
drivers_RT_RT700_MIMXRT798S.flexio_i2s_edma_handle_t (*C type*), 356
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_Enable (*C function*), 377
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_EnableInterrupts (*C function*), 377
drivers_RT_RT700_MIMXRT798S.flexio_i2s_format_t (*C type*), 385
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_GetDefaultConfig (*C function*), 377
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_GetStatusFlags (*C function*), 377
drivers_RT_RT700_MIMXRT798S.flexio_i2s_handle_t (*C type*), 385
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_Init (*C function*), 376

drivers_RT_RT700_MIMXRT798S.flexio_i2s_master_slave_t (*C type*), 385
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_MasterSetFormat (*C function*), 378
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_ReadBlocking (*C function*), 379
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_ReadData (*C function*), 380
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_RxEnableDMA (*C function*), 378
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_RxGetDataRegisterAddress (*C function*), 378
drivers_RT_RT700_MIMXRT798S.flexio_i2s_sample_rate_t (*C type*), 385
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_SlaveSetFormat (*C function*), 378
drivers_RT_RT700_MIMXRT798S.flexio_i2s_transfer_t (*C type*), 385
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferAbortReceive (*C function*), 382
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferAbortReceiveEDMA (*C function*), 356
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferAbortSend (*C function*), 381
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferAbortSendEDMA (*C function*), 356
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferGetReceiveCount (*C function*), 382
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferGetReceiveCountEDMA (*C function*), 356
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferGetSendCount (*C function*), 382
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferGetSendCountEDMA (*C function*), 356
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferReceiveEDMA (*C function*), 355
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferReceiveNonBlocking (*C function*), 381
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferRxCreateHandle (*C function*), 380
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferRxCreateHandleEDMA (*C function*), 354
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferRxHandleIRQ (*C function*), 383
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferSendEDMA (*C function*), 355
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferSendNonBlocking (*C function*), 380
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferSetFormat (*C function*), 380
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferSetFormatEDMA (*C function*), 355
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferTxCreateHandle (*C function*), 380
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferTxCreateHandleEDMA (*C function*), 354
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TransferTxHandleIRQ (*C function*), 382
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TxEnableDMA (*C function*), 378
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_TxGetDataRegisterAddress (*C function*), 378
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_Type (*C type*), 385
drivers_RT_RT700_MIMXRT798S.flexio_i2s_word_width_t (*C type*), 385
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_WriteBlocking (*C function*), 379
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_WriteData (*C function*), 379
drivers_RT_RT700_MIMXRT798S.FLEXIO_I2S_XFER_QUEUE_SIZE (*C macro*), 385
drivers_RT_RT700_MIMXRT798S.FLEXIO_Init (*C function*), 334
drivers_RT_RT700_MIMXRT798S.flexio_isr_t (*C type*), 350
drivers_RT_RT700_MIMXRT798S.flexio_mculcd_bus_t (*C type*), 398
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_ClearMultiBeatsReadConfig (*C function*), 392
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_ClearMultiBeatsWriteConfig (*C function*), 392
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_ClearSingleBeatReadConfig (*C function*), 391
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_ClearSingleBeatWriteConfig (*C function*), 391
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_ClearStatusFlags (*C function*), 389
drivers_RT_RT700_MIMXRT798S.flexio_mculcd_config_t (*C type*), 398
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_DATA_BUS_WIDTH (*C macro*), 398
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_Deinit (*C function*), 388
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_DisableInterrupts (*C function*), 389
drivers_RT_RT700_MIMXRT798S.flexio_mculcd_edma_handle_t (*C type*), 359
drivers_RT_RT700_MIMXRT798S.flexio_mculcd_edma_transfer_callback_t (*C type*), 359
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_Enable (*C function*), 392
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_EnableInterrupts (*C function*), 389
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_EnableRxDMA (*C function*), 390
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_EnableTxDMA (*C function*), 389

drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_GetDefaultConfig (*C function*), 389
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_GetRxDataRegisterAddress (*C function*), 390
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_GetStatusFlags (*C function*), 389
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_GetTxDataRegisterAddress (*C function*), 390
drivers_RT_RT700_MIMXRT798S.flexio_mculcd_handle_t (*C type*), 398
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_Init (*C function*), 388
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_LEGACY_GPIO_FUNC (*C macro*), 398
drivers_RT_RT700_MIMXRT798S.flexio_mculcd_pin_func_t (*C type*), 398
drivers_RT_RT700_MIMXRT798S.flexio_mculcd_pixel_format_t (*C type*), 397
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_ReadData (*C function*), 392
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_ReadDataArrayBlocking (*C function*), 394
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_SetBaudRate (*C function*), 390
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_SetMultiBeatsReadConfig (*C function*), 392
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_SetMultiBeatsWriteConfig (*C function*), 391
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_SetSingleBeatReadConfig (*C function*), 391
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_SetSingleBeatWriteConfig (*C function*), 390
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_StartTransfer (*C function*), 393
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_StopTransfer (*C function*), 393
drivers_RT_RT700_MIMXRT798S.flexio_mculcd_transfer_callback_t (*C type*), 398
drivers_RT_RT700_MIMXRT798S.flexio_mculcd_transfer_mode_t (*C type*), 398
drivers_RT_RT700_MIMXRT798S.flexio_mculcd_transfer_t (*C type*), 398
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_TransferAbort (*C function*), 395
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_TransferAbortEDMA (*C function*), 358
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_TransferBlocking (*C function*), 394
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_TransferCreateHandle (*C function*), 394
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_TransferCreateHandleEDMA (*C function*), 358
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_TransferEDMA (*C function*), 358
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_TransferGetCount (*C function*), 395
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_TransferGetCountEDMA (*C function*), 359
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_TransferHandleIRQ (*C function*), 396
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_TransferNonBlocking (*C function*), 395
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_Type (*C type*), 398
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_WAIT_COMPLETE_TIME (*C macro*), 398
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_WaitTransmitComplete (*C function*), 393
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_WriteCommandBlocking (*C function*), 393
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_WriteData (*C function*), 393
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_WriteDataArrayBlocking (*C function*), 394
drivers_RT_RT700_MIMXRT798S.FLEXIO_MCULCD_WriteSameValueBlocking (*C function*), 394
drivers_RT_RT700_MIMXRT798S.flexio_pin_config_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.flexio_pin_input_config_t (*C type*), 350
drivers_RT_RT700_MIMXRT798S.flexio_pin_polarity_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.FLEXIO_PinRead (*C function*), 341
drivers_RT_RT700_MIMXRT798S.FLEXIO_PinWrite (*C function*), 340
drivers_RT_RT700_MIMXRT798S.FLEXIO_ReadPinInput (*C function*), 335
drivers_RT_RT700_MIMXRT798S.FLEXIO_RegisterHandleIRQ (*C function*), 340
drivers_RT_RT700_MIMXRT798S.FLEXIO_Reset (*C function*), 335
drivers_RT_RT700_MIMXRT798S.FLEXIO_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.FLEXIO_SetClockMode (*C function*), 336
drivers_RT_RT700_MIMXRT798S.FLEXIO_SetPinConfig (*C function*), 351
drivers_RT_RT700_MIMXRT798S.FLEXIO_SetPinLevel (*C function*), 341
drivers_RT_RT700_MIMXRT798S.FLEXIO_SetPortOutput (*C function*), 340
drivers_RT_RT700_MIMXRT798S.FLEXIO_SetShifterConfig (*C function*), 335
drivers_RT_RT700_MIMXRT798S.FLEXIO_SetTimerConfig (*C function*), 336
drivers_RT_RT700_MIMXRT798S.flexio_shifter_buffer_type_t (*C type*), 350

drivers_RT_RT700_MIMXRT798S.flexio_shifter_config_t (*C type*), 350
drivers_RT_RT700_MIMXRT798S.flexio_shifter_input_source_t (*C type*), 350
drivers_RT_RT700_MIMXRT798S.flexio_shifter_mode_t (*C type*), 350
drivers_RT_RT700_MIMXRT798S.flexio_shifter_start_bit_t (*C type*), 350
drivers_RT_RT700_MIMXRT798S.flexio_shifter_stop_bit_t (*C type*), 350
drivers_RT_RT700_MIMXRT798S.flexio_shifter_timer_polarity_t (*C type*), 350
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_ClearStatusFlags (*C function*), 403
drivers_RT_RT700_MIMXRT798S.flexio_spi_clock_phase_t (*C type*), 411
drivers_RT_RT700_MIMXRT798S.flexio_spi_data_bitcount_mode_t (*C type*), 411
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_DisableInterrupts (*C function*), 404
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_DUMMYDATA (*C macro*), 412
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_Enable (*C function*), 404
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_EnableDMA (*C function*), 404
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_EnableInterrupts (*C function*), 403
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_FlushShifters (*C function*), 406
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_GetRxDataRegisterAddress (*C function*), 404
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_GetStatusFlags (*C function*), 403
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_GetTxDataRegisterAddress (*C function*), 404
drivers_RT_RT700_MIMXRT798S.flexio_spi_master_config_t (*C type*), 412
drivers_RT_RT700_MIMXRT798S.flexio_spi_master_edma_handle_t (*C type*), 363
drivers_RT_RT700_MIMXRT798S.flexio_spi_master_edma_transfer_callback_t (*C type*), 363
drivers_RT_RT700_MIMXRT798S.flexio_spi_master_handle_t (*C type*), 412
drivers_RT_RT700_MIMXRT798S.flexio_spi_master_transfer_callback_t (*C type*), 412
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterDeinit (*C function*), 402
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterGetDefaultConfig (*C function*), 402
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterInit (*C function*), 401
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterSetBaudRate (*C function*), 405
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterTransferAbort (*C function*), 407
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterTransferAbortEDMA (*C function*), 361
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterTransferBlocking (*C function*), 406
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterTransferCreateHandle (*C function*), 406
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterTransferCreateHandleEDMA (*C function*), 360
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterTransferEDMA (*C function*), 361
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterTransferGetCount (*C function*), 407
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterTransferGetCountEDMA (*C function*), 361
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterTransferHandleIRQ (*C function*), 407
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_MasterTransferNonBlocking (*C function*), 407
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_ReadBlocking (*C function*), 406
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_ReadData (*C function*), 405
drivers_RT_RT700_MIMXRT798S.flexio_spi_shift_direction_t (*C type*), 411
drivers_RT_RT700_MIMXRT798S.flexio_spi_slave_config_t (*C type*), 412
drivers_RT_RT700_MIMXRT798S.flexio_spi_slave_edma_handle_t (*C type*), 363
drivers_RT_RT700_MIMXRT798S.flexio_spi_slave_edma_transfer_callback_t (*C type*), 363
drivers_RT_RT700_MIMXRT798S.flexio_spi_slave_handle_t (*C type*), 412
drivers_RT_RT700_MIMXRT798S.flexio_spi_slave_transfer_callback_t (*C type*), 412
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_SlaveDeinit (*C function*), 403
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_SlaveGetDefaultConfig (*C function*), 403
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_SlaveInit (*C function*), 402
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_SlaveTransferAbort (*C function*), 408
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_SlaveTransferAbortEDMA (*C function*), 362
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_SlaveTransferCreateHandle (*C function*), 408
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_SlaveTransferCreateHandleEDMA (*C function*), 362
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_SlaveTransferEDMA (*C function*), 362
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_SlaveTransferGetCount (*C function*), 408
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_SlaveTransferGetCountEDMA (*C function*), 362
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_SlaveTransferHandleIRQ (*C function*), 409

drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_SlaveTransferNonBlocking (*C function*), 408
drivers_RT_RT700_MIMXRT798S.flexio_spi_transfer_t (*C type*), 412
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_Type (*C type*), 412
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_WriteBlocking (*C function*), 405
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_WriteData (*C function*), 405
drivers_RT_RT700_MIMXRT798S.FLEXIO_SPI_XFER_DATA_FORMAT (*C macro*), 412
drivers_RT_RT700_MIMXRT798S.flexio_timer_config_t (*C type*), 350
drivers_RT_RT700_MIMXRT798S.flexio_timer_decrement_source_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.flexio_timer_disable_condition_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.flexio_timer_enable_condition_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.flexio_timer_mode_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.flexio_timer_output_state_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.flexio_timer_output_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.flexio_timer_reset_condition_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.flexio_timer_start_bit_condition_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.flexio_timer_stop_bit_condition_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.flexio_timer_trigger_polarity_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.FLEXIO_TIMER_TRIGGER_SEL_PININPUT (*C macro*), 351
drivers_RT_RT700_MIMXRT798S.FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT (*C macro*), 351
drivers_RT_RT700_MIMXRT798S.FLEXIO_TIMER_TRIGGER_SEL_TIMn (*C macro*), 351
drivers_RT_RT700_MIMXRT798S.flexio_timer_trigger_source_t (*C type*), 349
drivers_RT_RT700_MIMXRT798S.FLEXIO_TogglePortOutput (*C function*), 340
drivers_RT_RT700_MIMXRT798S.flexio_uart_bit_count_per_char_t (*C type*), 424
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_ClearStatusFlags (*C function*), 416
drivers_RT_RT700_MIMXRT798S.flexio_uart_config_t (*C type*), 424
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_Deinit (*C function*), 416
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_DisableInterrupts (*C function*), 417
drivers_RT_RT700_MIMXRT798S.flexio_uart_edma_handle_t (*C type*), 366
drivers_RT_RT700_MIMXRT798S.flexio_uart_edma_transfer_callback_t (*C type*), 366
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_Enable (*C function*), 418
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_EnableInterrupts (*C function*), 417
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_EnableRxDMA (*C function*), 417
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_EnableTxDMA (*C function*), 417
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_FlushShifters (*C function*), 422
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_GetDefaultConfig (*C function*), 416
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_GetRxDataRegisterAddress (*C function*), 417
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_GetStatusFlags (*C function*), 416
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_GetTxDataRegisterAddress (*C function*), 417
drivers_RT_RT700_MIMXRT798S.flexio_uart_handle_t (*C type*), 424
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_Init (*C function*), 415
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_ReadBlocking (*C function*), 418
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_ReadByte (*C function*), 418
drivers_RT_RT700_MIMXRT798S.flexio_uart_transfer_callback_t (*C type*), 424
drivers_RT_RT700_MIMXRT798S.flexio_uart_transfer_t (*C type*), 424
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferAbortReceive (*C function*), 422
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferAbortReceiveEDMA (*C function*), 365
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferAbortSend (*C function*), 420
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferAbortSendEDMA (*C function*), 365
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferCreateHandle (*C function*), 419
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferCreateHandleEDMA (*C function*), 364
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferGetReceiveCount (*C function*), 422
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferGetReceiveCountEDMA (*C function*), 365
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferGetSendCount (*C function*), 421
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferGetSendCountEDMA (*C function*), 365
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferHandleIRQ (*C function*), 422
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferReceiveEDMA (*C function*), 364

drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferReceiveNonBlocking (*C function*), 421
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferSendEDMA (*C function*), 364
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferSendNonBlocking (*C function*), 420
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferStartRingBuffer (*C function*), 419
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_TransferStopRingBuffer (*C function*), 420
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_Type (*C type*), 424
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_WriteBlocking (*C function*), 418
drivers_RT_RT700_MIMXRT798S.FLEXIO_UART_WriteByte (*C function*), 418
drivers_RT_RT700_MIMXRT798S.FLEXIO_UnregisterHandleIRQ (*C function*), 340
drivers_RT_RT700_MIMXRT798S.FLEXIOCLKDIV_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.FLEXIOCLKSEL_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.freq_measure_config_t (*C type*), 609
drivers_RT_RT700_MIMXRT798S.FREQME_CalculateTargetClkFreq (*C function*), 606
drivers_RT_RT700_MIMXRT798S.FREQME_CheckContinuousMode (*C function*), 605
drivers_RT_RT700_MIMXRT798S.FREQME_CheckOperateMode (*C function*), 606
drivers_RT_RT700_MIMXRT798S.FREQME_CheckPulsePolarity (*C function*), 607
drivers_RT_RT700_MIMXRT798S.FREQME_ClearInterruptStatusFlags (*C function*), 607
drivers_RT_RT700_MIMXRT798S.FREQME_CLOCKS (*C macro*), 258
drivers_RT_RT700_MIMXRT798S.FREQME_DisableInterrupts (*C function*), 608
drivers_RT_RT700_MIMXRT798S.FREQME_EnableContinuousMode (*C function*), 605
drivers_RT_RT700_MIMXRT798S.FREQME_EnableInterrupts (*C function*), 608
drivers_RT_RT700_MIMXRT798S.FREQME_GetDefaultConfig (*C function*), 605
drivers_RT_RT700_MIMXRT798S.FREQME_GetInterruptStatusFlags (*C function*), 607
drivers_RT_RT700_MIMXRT798S.FREQME_GetMeasurementResult (*C function*), 607
drivers_RT_RT700_MIMXRT798S.FREQME_GetReferenceClkScaleValue (*C function*), 607
drivers_RT_RT700_MIMXRT798S.FREQME_Init (*C function*), 604
drivers_RT_RT700_MIMXRT798S.freqme_mode_attribute_t (*C type*), 609
drivers_RT_RT700_MIMXRT798S.freqme_operate_mode_t (*C type*), 609
drivers_RT_RT700_MIMXRT798S.freqme_pulse_polarity_t (*C type*), 609
drivers_RT_RT700_MIMXRT798S.FREQME_RSTS_N (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.FREQME_SetMaxExpectedValue (*C function*), 606
drivers_RT_RT700_MIMXRT798S.FREQME_SetMinExpectedValue (*C function*), 606
drivers_RT_RT700_MIMXRT798S.FREQME_SetOperateMode (*C function*), 605
drivers_RT_RT700_MIMXRT798S.FREQME_SetPulsePolarity (*C function*), 607
drivers_RT_RT700_MIMXRT798S.FREQME_SetReferenceClk (*C function*), 606
drivers_RT_RT700_MIMXRT798S.FREQME_SetTargetClk (*C function*), 606
drivers_RT_RT700_MIMXRT798S.FREQME_StartMeasurementCycle (*C function*), 605
drivers_RT_RT700_MIMXRT798S.FREQME_TerminateMeasurementCycle (*C function*), 605
drivers_RT_RT700_MIMXRT798S.FSL_ACMP_DRIVER_VERSION (*C macro*), 198
drivers_RT_RT700_MIMXRT798S.FSL_CACHE_DRIVER_VERSION (*C macro*), 205, 980
drivers_RT_RT700_MIMXRT798S.FSL_CDOG_DRIVER_VERSION (*C macro*), 209
drivers_RT_RT700_MIMXRT798S.FSL_CLOCK_DRIVER_VERSION (*C macro*), 258
drivers_RT_RT700_MIMXRT798S.FSL_COMMON_DRIVER_VERSION (*C macro*), 538
drivers_RT_RT700_MIMXRT798S.FSL_COMPONENT_ID (*C macro*), 516, 928
drivers_RT_RT700_MIMXRT798S.FSL_CRC_DRIVER_VERSION (*C macro*), 267
drivers_RT_RT700_MIMXRT798S.FSL_CTIMER_DRIVER_VERSION (*C macro*), 274
drivers_RT_RT700_MIMXRT798S.FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.FSL_DSI_RX_MAX_PAYLOAD_BYTE (*C macro*), 718
drivers_RT_RT700_MIMXRT798S.FSL_DSI_TX_MAX_PAYLOAD_BYTE (*C macro*), 718
drivers_RT_RT700_MIMXRT798S.FSL_DSP_DRIVER_VERSION (*C macro*), 282
drivers_RT_RT700_MIMXRT798S.FSL_EDMA_DRIVER_EDMA4 (*C macro*), 311
drivers_RT_RT700_MIMXRT798S.FSL_EDMA_DRIVER_UNIFIED (*C macro*), 311
drivers_RT_RT700_MIMXRT798S.FSL_EDMA_DRIVER_VERSION (*C macro*), 301
drivers_RT_RT700_MIMXRT798S.FSL_EDMA_SOC_DRIVER_VERSION (*C macro*), 326
drivers_RT_RT700_MIMXRT798S.FSL_EDMA_SOC_IP_DMA3 (*C macro*), 326
drivers_RT_RT700_MIMXRT798S.FSL_EDMA_SOC_IP_DMA4 (*C macro*), 326
drivers_RT_RT700_MIMXRT798S.FSL_EZHV_DRIVER_VERSION (*C macro*), 329

drivers_RT_RT700_MIMXRT798S.FSL_FLEXIO_CAMERA_DRIVER_VERSION (*C macro*), 332
drivers_RT_RT700_MIMXRT798S.FSL_FLEXIO_DRIVER_VERSION (*C macro*), 342
drivers_RT_RT700_MIMXRT798S.FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION (*C macro*), 373
drivers_RT_RT700_MIMXRT798S.FSL_FLEXIO_I2S_DRIVER_VERSION (*C macro*), 383
drivers_RT_RT700_MIMXRT798S.FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION (*C macro*), 356
drivers_RT_RT700_MIMXRT798S.FSL_FLEXIO_MCULCD_DRIVER_VERSION (*C macro*), 396
drivers_RT_RT700_MIMXRT798S.FSL_FLEXIO_MCULCD_EDMA_DRIVER_VERSION (*C macro*), 359
drivers_RT_RT700_MIMXRT798S.FSL_FLEXIO_SPI_DRIVER_VERSION (*C macro*), 409
drivers_RT_RT700_MIMXRT798S.FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION (*C macro*), 363
drivers_RT_RT700_MIMXRT798S.FSL_FLEXIO_UART_DRIVER_VERSION (*C macro*), 422
drivers_RT_RT700_MIMXRT798S.FSL_FLEXIO_UART_EDMA_DRIVER_VERSION (*C macro*), 366
drivers_RT_RT700_MIMXRT798S.FSL_FREQME_DRIVER_VERSION (*C macro*), 608
drivers_RT_RT700_MIMXRT798S.FSL_GDET_DRIVER_VERSION (*C macro*), 428
drivers_RT_RT700_MIMXRT798S.FSL_GLIKEY_DRIVER_VERSION (*C macro*), 429
drivers_RT_RT700_MIMXRT798S.FSL_GPIO_DRIVER_VERSION (*C macro*), 434
drivers_RT_RT700_MIMXRT798S.FSL_I3C_DRIVER_VERSION (*C macro*), 443
drivers_RT_RT700_MIMXRT798S.FSL_INPUTMUX_DRIVER_VERSION (*C macro*), 512
drivers_RT_RT700_MIMXRT798S.FSL_IRTC_DRIVER_VERSION (*C macro*), 519
drivers_RT_RT700_MIMXRT798S.FSL_ITRC_DRIVER_VERSION (*C macro*), 527
drivers_RT_RT700_MIMXRT798S.FSL_JPEGDEC_DRIVER_VERSION (*C macro*), 532
drivers_RT_RT700_MIMXRT798S.FSL_LCDIF_DRIVER_VERSION (*C macro*), 566
drivers_RT_RT700_MIMXRT798S.FSL_LP_FLEXCOMM_DRIVER_VERSION (*C macro*), 610
drivers_RT_RT700_MIMXRT798S.FSL_LPADC_DRIVER_VERSION (*C macro*), 601
drivers_RT_RT700_MIMXRT798S.FSL_LPI2C_DRIVER_VERSION (*C macro*), 612
drivers_RT_RT700_MIMXRT798S.FSL_LPSPI_DRIVER_VERSION (*C macro*), 652
drivers_RT_RT700_MIMXRT798S.FSL_LPSPI_EDMA_DRIVER_VERSION (*C macro*), 666
drivers_RT_RT700_MIMXRT798S.FSL_LPUART_DRIVER_VERSION (*C macro*), 683
drivers_RT_RT700_MIMXRT798S.FSL_LPUART_EDMA_DRIVER_VERSION (*C macro*), 698
drivers_RT_RT700_MIMXRT798S.FSL_MIPI_DSI_DRIVER_VERSION (*C macro*), 706
drivers_RT_RT700_MIMXRT798S.FSL_MMU_DRIVER_VERSION (*C macro*), 729
drivers_RT_RT700_MIMXRT798S.FSL_MRT_DRIVER_VERSION (*C macro*), 737
drivers_RT_RT700_MIMXRT798S.FSL_MU_DRIVER_VERSION (*C macro*), 748
drivers_RT_RT700_MIMXRT798S.FSL_OSTIMER_DRIVER_VERSION (*C macro*), 758
drivers_RT_RT700_MIMXRT798S.FSL_PDM_DRIVER_VERSION (*C macro*), 764
drivers_RT_RT700_MIMXRT798S.FSL_PDM_EDMA_DRIVER_VERSION (*C macro*), 774
drivers_RT_RT700_MIMXRT798S.FSL_PINT_DRIVER_VERSION (*C macro*), 776
drivers_RT_RT700_MIMXRT798S.FSL_PNGDEC_DRIVER_VERSION (*C macro*), 787
drivers_RT_RT700_MIMXRT798S.FSL_POWER_DRIVER_VERSION (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.FSL_PUF_V3_DRIVER_VERSION (*C macro*), 820
drivers_RT_RT700_MIMXRT798S.FSL_PVTS_DRIVER_VERSION (*C macro*), 828
drivers_RT_RT700_MIMXRT798S.FSL_RESET_DRIVER_VERSION (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.FSL_SAI_DRIVER_VERSION (*C macro*), 857
drivers_RT_RT700_MIMXRT798S.FSL_SAI_EDMA_DRIVER_VERSION (*C macro*), 877
drivers_RT_RT700_MIMXRT798S.FSL_SAI_HAS_FIFO_EXTEND_FEATURE (*C macro*), 866
drivers_RT_RT700_MIMXRT798S.FSL_SCTIMER_DRIVER_VERSION (*C macro*), 897
drivers_RT_RT700_MIMXRT798S.FSL_SDADC_DRIVER_VERSION (*C macro*), 915
drivers_RT_RT700_MIMXRT798S.FSL_SEMA42_DRIVER_VERSION (*C macro*), 924
drivers_RT_RT700_MIMXRT798S.FSL_SOC_MIPI_DSI_DRIVER_VERSION (*C macro*), 928
drivers_RT_RT700_MIMXRT798S.FSL_SYSPM_DRIVER_VERSION (*C macro*), 931
drivers_RT_RT700_MIMXRT798S.FSL_TRNG_DRIVER_VERSION (*C macro*), 931
drivers_RT_RT700_MIMXRT798S.FSL_USDHC_DRIVER_VERSION (*C macro*), 948
drivers_RT_RT700_MIMXRT798S.FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER (*C macro*), 964

drivers_RT_RT700_MIMXRT798S.FSL_UTICK_DRIVER_VERSION (*C macro*), 973
drivers_RT_RT700_MIMXRT798S.FSL_WWDT_DRIVER_VERSION (*C macro*), 976
drivers_RT_RT700_MIMXRT798S.FSL_XSPI_DRIVER_VERSION (*C macro*), 1003
drivers_RT_RT700_MIMXRT798S.FSL_XSPI_EDMA_DRIVER_VERSION (*C macro*), 1049
drivers_RT_RT700_MIMXRT798S.fw_ver_check (*C struct*), 882
drivers_RT_RT700_MIMXRT798S.g_32kClkinFreq (*C var*), 247
drivers_RT_RT700_MIMXRT798S.g_clkinFreq (*C var*), 247
drivers_RT_RT700_MIMXRT798S.g_lpspiDummyData (*C var*), 661
drivers_RT_RT700_MIMXRT798S.g_mclkFreq (*C var*), 247
drivers_RT_RT700_MIMXRT798S.g_selectedPageOpt (*C var*), 732
drivers_RT_RT700_MIMXRT798S.g_selectedUnitSize (*C var*), 732
drivers_RT_RT700_MIMXRT798S.g_xtalFreq (*C var*), 247
drivers_RT_RT700_MIMXRT798S.gdet_core_voltage_t (*C type*), 429
drivers_RT_RT700_MIMXRT798S.GDET_Deinit (*C function*), 427
drivers_RT_RT700_MIMXRT798S.GDET_Disable (*C function*), 427
drivers_RT_RT700_MIMXRT798S.GDET_DriverIRQHandler (*C function*), 429
drivers_RT_RT700_MIMXRT798S.GDET_Enable (*C function*), 427
drivers_RT_RT700_MIMXRT798S.GDET_Init (*C function*), 427
drivers_RT_RT700_MIMXRT798S.GDET_IsolateOff (*C function*), 428
drivers_RT_RT700_MIMXRT798S.GDET_IsolateOn (*C function*), 427
drivers_RT_RT700_MIMXRT798S.GDET_ReconfigureVoltageMode (*C function*), 428
drivers_RT_RT700_MIMXRT798S.GET_LP_CTRL_BIT_FROM_BITS (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.GET_LP_CTRL_REG_FROM_BITS (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.GET_LP_STATE_BIT_FROM_BITS (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.GET_LP_STATE_REG_FROM_BITS (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.GET_PD_REG_FROM_BITS (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.GLIKEY_CheckLock (*C function*), 431
drivers_RT_RT700_MIMXRT798S.GLIKEY_ClearIntStatus (*C function*), 431
drivers_RT_RT700_MIMXRT798S.GLIKEY_CODEWORD_STEP1 (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_CODEWORD_STEP2 (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_CODEWORD_STEP3 (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_CODEWORD_STEP4 (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_CODEWORD_STEP5 (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_CODEWORD_STEP6 (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_CODEWORD_STEP7 (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_CODEWORD_STEP_EN (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_ContinueEnable (*C function*), 433
drivers_RT_RT700_MIMXRT798S.GLIKEY_EndOperation (*C function*), 433
drivers_RT_RT700_MIMXRT798S.GLIKEY_FSM_INIT (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_FSM_LOCKED (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_FSM_SSR_RESET (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_FSM_STEP1 (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_FSM_STEP2 (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_FSM_STEP3 (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_FSM_STEP4 (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_FSM_WR_DIS (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_FSM_WR_EN (*C macro*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_GetIntEnable (*C function*), 431
drivers_RT_RT700_MIMXRT798S.GLIKEY_GetStatus (*C function*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_IsIndexLocked (*C function*), 432
drivers_RT_RT700_MIMXRT798S.GLIKEY_IsLocked (*C function*), 430
drivers_RT_RT700_MIMXRT798S.GLIKEY_Lock (*C function*), 432
drivers_RT_RT700_MIMXRT798S.GLIKEY_LockIndex (*C function*), 432
drivers_RT_RT700_MIMXRT798S.GLIKEY_ResetIndex (*C function*), 433
drivers_RT_RT700_MIMXRT798S.GLIKEY_SetIntEnable (*C function*), 431
drivers_RT_RT700_MIMXRT798S.GLIKEY_SetIntStatus (*C function*), 432
drivers_RT_RT700_MIMXRT798S.GLIKEY_StartEnable (*C function*), 432
drivers_RT_RT700_MIMXRT798S.GLIKEY_SyncReset (*C function*), 431

drivers_RT_RT700_MIMXRT798S.GPIO_CheckAttributeBytes (*C function*), 443
drivers_RT_RT700_MIMXRT798S.gpio_checker_attribute_t (*C type*), 436
drivers_RT_RT700_MIMXRT798S.GPIO_CLOCKS (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.GPIO_DisableInterruptControlNonPrivilege (*C function*), 439
drivers_RT_RT700_MIMXRT798S.GPIO_DisableInterruptControlNonSecure (*C function*), 439
drivers_RT_RT700_MIMXRT798S.GPIO_DisablePinControlNonPrivilege (*C function*), 439
drivers_RT_RT700_MIMXRT798S.GPIO_DisablePinControlNonSecure (*C function*), 438
drivers_RT_RT700_MIMXRT798S.GPIO_EnableInterruptControlNonPrivilege (*C function*), 439
drivers_RT_RT700_MIMXRT798S.GPIO_EnableInterruptControlNonSecure (*C function*), 439
drivers_RT_RT700_MIMXRT798S.GPIO_EnablePinControlNonPrivilege (*C function*), 439
drivers_RT_RT700_MIMXRT798S.GPIO_EnablePinControlNonSecure (*C function*), 438
drivers_RT_RT700_MIMXRT798S.GPIO_FIT_REG (*C macro*), 437
drivers_RT_RT700_MIMXRT798S.GPIO_GetVersionInfo (*C function*), 438
drivers_RT_RT700_MIMXRT798S.GPIO_GpioClearInterruptChannelFlags (*C function*), 442
drivers_RT_RT700_MIMXRT798S.GPIO_GpioClearInterruptFlags (*C function*), 442
drivers_RT_RT700_MIMXRT798S.GPIO_GpioGetInterruptChannelFlags (*C function*), 441
drivers_RT_RT700_MIMXRT798S.GPIO_GpioGetInterruptFlags (*C function*), 441
drivers_RT_RT700_MIMXRT798S.gpio_interrupt_config_t (*C type*), 436
drivers_RT_RT700_MIMXRT798S.gpio_interrupt_selection_t (*C type*), 436
drivers_RT_RT700_MIMXRT798S.gpio_pin_config_t (*C type*), 436
drivers_RT_RT700_MIMXRT798S.gpio_pin_direction_t (*C type*), 436
drivers_RT_RT700_MIMXRT798S.gpio_pin_interrupt_control_t (*C enum*), 436
drivers_RT_RT700_MIMXRT798S.gpio_pin_interrupt_control_t.kGPIO_InterruptControlNonPrivilege (*C enumerator*), 436
drivers_RT_RT700_MIMXRT798S.gpio_pin_interrupt_control_t.kGPIO_InterruptControlNonSecure (*C enumerator*), 436
drivers_RT_RT700_MIMXRT798S.gpio_pin_interrupt_control_t.kGPIO_PinControlNonPrivilege (*C enumerator*), 436
drivers_RT_RT700_MIMXRT798S.gpio_pin_interrupt_control_t.kGPIO_PinControlNonSecure (*C enumerator*), 436
drivers_RT_RT700_MIMXRT798S.GPIO_PinClearInterruptFlag (*C function*), 442
drivers_RT_RT700_MIMXRT798S.GPIO_PinGetInterruptFlag (*C function*), 442
drivers_RT_RT700_MIMXRT798S.GPIO_PinInit (*C function*), 438
drivers_RT_RT700_MIMXRT798S.GPIO_PinRead (*C function*), 440
drivers_RT_RT700_MIMXRT798S.GPIO_PinWrite (*C function*), 440
drivers_RT_RT700_MIMXRT798S.GPIO_PortClear (*C function*), 440
drivers_RT_RT700_MIMXRT798S.GPIO_PortClearInterruptFlags (*C macro*), 443
drivers_RT_RT700_MIMXRT798S.GPIO_PortDenit (*C function*), 437
drivers_RT_RT700_MIMXRT798S.GPIO_PortGetInterruptFlags (*C macro*), 443
drivers_RT_RT700_MIMXRT798S.GPIO_PortInit (*C function*), 437
drivers_RT_RT700_MIMXRT798S.GPIO_PortInputDisable (*C function*), 439
drivers_RT_RT700_MIMXRT798S.GPIO_PortInputEnable (*C function*), 439
drivers_RT_RT700_MIMXRT798S.GPIO_PortSet (*C function*), 440
drivers_RT_RT700_MIMXRT798S.GPIO_PortToggle (*C function*), 440
drivers_RT_RT700_MIMXRT798S.GPIO_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.GPIO_SecurePrivilegeLock (*C function*), 438
drivers_RT_RT700_MIMXRT798S.GPIO_SetMultipleInterruptPinsConfig (*C function*), 442
drivers_RT_RT700_MIMXRT798S.GPIO_SetPinInterruptChannel (*C function*), 441
drivers_RT_RT700_MIMXRT798S.GPIO_SetPinInterruptConfig (*C function*), 440
drivers_RT_RT700_MIMXRT798S.gpio_version_info_t (*C type*), 437
drivers_RT_RT700_MIMXRT798S.GPU_CLOCKS (*C macro*), 258
drivers_RT_RT700_MIMXRT798S.height (*C var*), 537, 790
drivers_RT_RT700_MIMXRT798S.I2C_RETRY_TIMES (*C macro*), 374, 612
drivers_RT_RT700_MIMXRT798S.I2S_RETRY_TIMES (*C macro*), 385
drivers_RT_RT700_MIMXRT798S.I3C23FCLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.I3C23FCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.i3c_baudrate_hz_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.i3c_bus_request_t (*C type*), 474

drivers_RT_RT700_MIMXRT798S.i3c_bus_type_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.I3C_CheckForBusyBus (*C function*), 450
drivers_RT_RT700_MIMXRT798S.I3C_CLOCKS (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.i3c_config_t (*C type*), 446
drivers_RT_RT700_MIMXRT798S.i3c_device_info_t (*C type*), 445
drivers_RT_RT700_MIMXRT798S.i3c_direction_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.I3C_DriverIRQHandler (*C function*), 466
drivers_RT_RT700_MIMXRT798S.I3C_GetDefaultConfig (*C function*), 446
drivers_RT_RT700_MIMXRT798S.I3C_GetIBIAddress (*C function*), 462
drivers_RT_RT700_MIMXRT798S.I3C_GetIBIType (*C function*), 462
drivers_RT_RT700_MIMXRT798S.I3C_GetInstance (*C function*), 446
drivers_RT_RT700_MIMXRT798S.i3c_hdr_mode_t (*C type*), 445
drivers_RT_RT700_MIMXRT798S.I3C_IBI_BUFF_SIZE (*C macro*), 445
drivers_RT_RT700_MIMXRT798S.i3c_ibi_response_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.i3c_ibi_state_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.i3c_ibi_type_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.I3C_Init (*C function*), 447
drivers_RT_RT700_MIMXRT798S.i3c_legacy_i2c_speed_mode_t (*C type*), 475
drivers_RT_RT700_MIMXRT798S.i3c_master_baudrate_config_t (*C type*), 475
drivers_RT_RT700_MIMXRT798S.i3c_master_bus_mode_t (*C type*), 475
drivers_RT_RT700_MIMXRT798S.i3c_master_config_t (*C type*), 475
drivers_RT_RT700_MIMXRT798S.i3c_master_daa_baudrate_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.i3c_master_edma_callback_t (*C type*), 482
drivers_RT_RT700_MIMXRT798S.i3c_master_edma_handle_t (*C type*), 482
drivers_RT_RT700_MIMXRT798S.i3c_master_enable_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.i3c_master_handle_t (*C type*), 475
drivers_RT_RT700_MIMXRT798S.i3c_master_hkeep_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.i3c_master_isr_t (*C type*), 475
drivers_RT_RT700_MIMXRT798S.i3c_master_src_clk_t (*C type*), 475
drivers_RT_RT700_MIMXRT798S.i3c_master_state_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.i3c_master_transfer_callback_t (*C type*), 475
drivers_RT_RT700_MIMXRT798S.i3c_master_transfer_t (*C type*), 475
drivers_RT_RT700_MIMXRT798S.I3C_MasterCalcTiming (*C function*), 458
drivers_RT_RT700_MIMXRT798S.I3C_MasterCheckAndClearError (*C function*), 449
drivers_RT_RT700_MIMXRT798S.I3C_MasterClearDeviceCount (*C function*), 463
drivers_RT_RT700_MIMXRT798S.I3C_MasterClearErrorStatusFlags (*C function*), 451
drivers_RT_RT700_MIMXRT798S.I3C_MasterClearStatusFlags (*C function*), 451
drivers_RT_RT700_MIMXRT798S.I3C_MasterDeinit (*C function*), 449
drivers_RT_RT700_MIMXRT798S.I3C_MasterDisableInterrupts (*C function*), 454
drivers_RT_RT700_MIMXRT798S.I3C_MasterEmitIBIResponse (*C function*), 462
drivers_RT_RT700_MIMXRT798S.I3C_MasterEmitRequest (*C function*), 462
drivers_RT_RT700_MIMXRT798S.I3C_MasterEnable (*C function*), 449
drivers_RT_RT700_MIMXRT798S.I3C_MasterEnableDMA (*C function*), 455
drivers_RT_RT700_MIMXRT798S.I3C_MasterEnableInterrupts (*C function*), 453
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetBusIdleState (*C function*), 458
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetDefaultConfig (*C function*), 449
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetDeviceListAfterDAA (*C function*), 463
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetEnabledInterrupts (*C function*), 454
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetErrorStatusFlags (*C function*), 451
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetFifoCounts (*C function*), 457
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetI2CPresetTiming (*C function*), 458
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetIBIRules (*C function*), 462
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetPendingInterrupts (*C function*), 454
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetRxFifoAddress (*C function*), 456
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetState (*C function*), 452
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetStatusFlags (*C function*), 450
drivers_RT_RT700_MIMXRT798S.I3C_MasterGetTxFifoAddress (*C function*), 456
drivers_RT_RT700_MIMXRT798S.I3C_MasterInit (*C function*), 449

drivers_RT_RT700_MIMXRT798S.I3C_MasterProcessDAA (*C function*), 463
drivers_RT_RT700_MIMXRT798S.I3C_MasterProcessDAASpecifiedBaudrate (*C function*), 462
drivers_RT_RT700_MIMXRT798S.I3C_MasterReceive (*C function*), 461
drivers_RT_RT700_MIMXRT798S.I3C_MasterRegisterIBI (*C function*), 462
drivers_RT_RT700_MIMXRT798S.I3C_MasterRepeatedStart (*C function*), 460
drivers_RT_RT700_MIMXRT798S.I3C_MasterRepeatedStartWithRxSize (*C function*), 459
drivers_RT_RT700_MIMXRT798S.I3C_MasterSend (*C function*), 460
drivers_RT_RT700_MIMXRT798S.I3C_MasterSetBaudRate (*C function*), 457
drivers_RT_RT700_MIMXRT798S.I3C_MasterSetTiming (*C function*), 458
drivers_RT_RT700_MIMXRT798S.I3C_MasterSetWatermarks (*C function*), 456
drivers_RT_RT700_MIMXRT798S.I3C_MasterStart (*C function*), 459
drivers_RT_RT700_MIMXRT798S.I3C_MasterStartWithRxSize (*C function*), 459
drivers_RT_RT700_MIMXRT798S.I3C_MasterStop (*C function*), 461
drivers_RT_RT700_MIMXRT798S.I3C_MasterTransferAbort (*C function*), 465
drivers_RT_RT700_MIMXRT798S.I3C_MasterTransferAbortEDMA (*C function*), 481
drivers_RT_RT700_MIMXRT798S.I3C_MasterTransferBlocking (*C function*), 463
drivers_RT_RT700_MIMXRT798S.I3C_MasterTransferCreateHandle (*C function*), 464
drivers_RT_RT700_MIMXRT798S.I3C_MasterTransferCreateHandleEDMA (*C function*), 480
drivers_RT_RT700_MIMXRT798S.I3C_MasterTransferEDMA (*C function*), 480
drivers_RT_RT700_MIMXRT798S.I3C_MasterTransferEDMAHandleIRQ (*C function*), 481
drivers_RT_RT700_MIMXRT798S.I3C_MasterTransferGetCount (*C function*), 465
drivers_RT_RT700_MIMXRT798S.I3C_MasterTransferGetCountEDMA (*C function*), 481
drivers_RT_RT700_MIMXRT798S.I3C_MasterTransferHandleIRQ (*C function*), 466
drivers_RT_RT700_MIMXRT798S.I3C_MasterTransferNonBlocking (*C function*), 465
drivers_RT_RT700_MIMXRT798S.I3C_MasterWaitForCtrlDone (*C function*), 450
drivers_RT_RT700_MIMXRT798S.I3C_MAX_DEVCNT (*C macro*), 445
drivers_RT_RT700_MIMXRT798S.i3c_register_ibi_addr_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.I3C_RETRY_TIMES (*C macro*), 445
drivers_RT_RT700_MIMXRT798S.I3C_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.i3c_rx_term_ops_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.i3c_rx_trigger_level_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.i3c_slave_activity_state_t (*C type*), 494
drivers_RT_RT700_MIMXRT798S.i3c_slave_config_t (*C type*), 494
drivers_RT_RT700_MIMXRT798S.i3c_slave_edma_callback_t (*C type*), 499
drivers_RT_RT700_MIMXRT798S.i3c_slave_edma_handle_t (*C type*), 499
drivers_RT_RT700_MIMXRT798S.i3c_slave_edma_transfer_t (*C type*), 499
drivers_RT_RT700_MIMXRT798S.i3c_slave_event_t (*C type*), 494
drivers_RT_RT700_MIMXRT798S.i3c_slave_handle_t (*C type*), 495
drivers_RT_RT700_MIMXRT798S.i3c_slave_isr_t (*C type*), 495
drivers_RT_RT700_MIMXRT798S.i3c_slave_transfer_callback_t (*C type*), 495
drivers_RT_RT700_MIMXRT798S.i3c_slave_transfer_event_t (*C type*), 494
drivers_RT_RT700_MIMXRT798S.i3c_slave_transfer_t (*C type*), 495
drivers_RT_RT700_MIMXRT798S.I3C_SlaveCheckAndClearError (*C function*), 453
drivers_RT_RT700_MIMXRT798S.I3C_SlaveClearErrorStatusFlags (*C function*), 453, 485
drivers_RT_RT700_MIMXRT798S.I3C_SlaveClearStatusFlags (*C function*), 452, 484
drivers_RT_RT700_MIMXRT798S.I3C_SlaveDeinit (*C function*), 450, 484
drivers_RT_RT700_MIMXRT798S.I3C_SlaveDisableInterrupts (*C function*), 455, 486
drivers_RT_RT700_MIMXRT798S.I3C_SlaveEnable (*C function*), 450, 484
drivers_RT_RT700_MIMXRT798S.I3C_SlaveEnableDMA (*C function*), 456, 487
drivers_RT_RT700_MIMXRT798S.I3C_SlaveEnableInterrupts (*C function*), 454, 485
drivers_RT_RT700_MIMXRT798S.I3C_SlaveGetActivityState (*C function*), 453, 485
drivers_RT_RT700_MIMXRT798S.I3C_SlaveGetDefaultConfig (*C function*), 450, 483
drivers_RT_RT700_MIMXRT798S.I3C_SlaveGetEnabledInterrupts (*C function*), 455, 486
drivers_RT_RT700_MIMXRT798S.I3C_SlaveGetErrorStatusFlags (*C function*), 453, 485
drivers_RT_RT700_MIMXRT798S.I3C_SlaveGetFifoCounts (*C function*), 457, 488
drivers_RT_RT700_MIMXRT798S.I3C_SlaveGetPendingInterrupts (*C function*), 455, 487
drivers_RT_RT700_MIMXRT798S.I3C_SlaveGetRxFifoAddress (*C function*), 456, 487
drivers_RT_RT700_MIMXRT798S.I3C_SlaveGetStatusFlags (*C function*), 452, 484

drivers_RT_RT700_MIMXRT798S.I3C_SlaveGetTxFifoAddress (*C function*), 456, 487
drivers_RT_RT700_MIMXRT798S.I3C_SlaveInit (*C function*), 450, 483
drivers_RT_RT700_MIMXRT798S.I3C_SlaveReceive (*C function*), 464, 488
drivers_RT_RT700_MIMXRT798S.I3C_SlaveSend (*C function*), 464, 488
drivers_RT_RT700_MIMXRT798S.I3C_SlaveSetWatermarks (*C function*), 457, 487
drivers_RT_RT700_MIMXRT798S.I3C_SlaveTransferAbort (*C function*), 489
drivers_RT_RT700_MIMXRT798S.I3C_SlaveTransferAbortEDMA (*C function*), 498
drivers_RT_RT700_MIMXRT798S.I3C_SlaveTransferCreateHandle (*C function*), 488
drivers_RT_RT700_MIMXRT798S.I3C_SlaveTransferCreateHandleEDMA (*C function*), 497
drivers_RT_RT700_MIMXRT798S.I3C_SlaveTransferEDMA (*C function*), 498
drivers_RT_RT700_MIMXRT798S.I3C_SlaveTransferEDMAHandleIRQ (*C function*), 498
drivers_RT_RT700_MIMXRT798S.I3C_SlaveTransferGetCount (*C function*), 489
drivers_RT_RT700_MIMXRT798S.I3C_SlaveTransferHandleIRQ (*C function*), 490
drivers_RT_RT700_MIMXRT798S.I3C_SlaveTransferNonBlocking (*C function*), 489
drivers_RT_RT700_MIMXRT798S.i3c_start_scl_delay_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.i3c_timing_t (*C type*), 475
drivers_RT_RT700_MIMXRT798S.i3c_tx_trigger_level_t (*C type*), 474
drivers_RT_RT700_MIMXRT798S.ICACHE_InvalidateByRange (*C function*), 204, 978
drivers_RT_RT700_MIMXRT798S.imageData (*C var*), 790
drivers_RT_RT700_MIMXRT798S.IN_0_15_EVENTS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.INPUTMUX_AttachSignal (*C function*), 512
drivers_RT_RT700_MIMXRT798S.inputmux_connection_t (*C type*), 512
drivers_RT_RT700_MIMXRT798S.INPUTMUX_Deinit (*C function*), 513
drivers_RT_RT700_MIMXRT798S.INPUTMUX_Init (*C function*), 512
drivers_RT_RT700_MIMXRT798S.instance (*C var*), 842
drivers_RT_RT700_MIMXRT798S.INSTRUCTION_COUNT_STABLE_TIMEOUT (*C macro*), 931
drivers_RT_RT700_MIMXRT798S.IOPCTL_ANAMUX_EN (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_DRIVE_33OHM (*C macro*), 516
drivers_RT_RT700_MIMXRT798S.IOPCTL_DRIVE_50OHM (*C macro*), 516
drivers_RT_RT700_MIMXRT798S.IOPCTL_DRIVE_66OHM (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_DRIVE_100OHM (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_FULLDRIVE_EN (*C macro*), 516
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC0 (*C macro*), 514
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC1 (*C macro*), 514
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC2 (*C macro*), 514
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC3 (*C macro*), 514
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC4 (*C macro*), 514
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC5 (*C macro*), 514
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC6 (*C macro*), 514
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC7 (*C macro*), 514
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC8 (*C macro*), 514
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC9 (*C macro*), 514
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC10 (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC11 (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC12 (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC13 (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC14 (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_FUNC15 (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.iopctl_group_t (*C type*), 513
drivers_RT_RT700_MIMXRT798S.IOPCTL_INBUF_EN (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_INV_EN (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_PSEDRAIN_EN (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_PULLDOWN_EN (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_PULLUP_EN (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_PUPD_EN (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IOPCTL_SLEW_RATE (*C macro*), 515
drivers_RT_RT700_MIMXRT798S.IRQ_ClearPendingIRQ (*C function*), 555
drivers_RT_RT700_MIMXRT798S.IRQ_SetPriority (*C function*), 555

drivers_RT_RT700_MIMXRT798S.irtc_alarm_match_t (*C type*), 522
drivers_RT_RT700_MIMXRT798S.IRTC_ClearStatusFlags (*C function*), 518
drivers_RT_RT700_MIMXRT798S.irtc_clock_select_t (*C type*), 522
drivers_RT_RT700_MIMXRT798S.irtc_clockout_sel_t (*C type*), 523
drivers_RT_RT700_MIMXRT798S.irtc_config_t (*C type*), 523
drivers_RT_RT700_MIMXRT798S.IRTC_ConfigClockOut (*C function*), 523
drivers_RT_RT700_MIMXRT798S.IRTC_ConfigClockSelect (*C function*), 523
drivers_RT_RT700_MIMXRT798S.irtc_datetime_t (*C type*), 523
drivers_RT_RT700_MIMXRT798S.irtc_daylight_time_t (*C type*), 523
drivers_RT_RT700_MIMXRT798S.IRTC_Deinit (*C function*), 516
drivers_RT_RT700_MIMXRT798S.IRTC_DisableInterrupts (*C function*), 518
drivers_RT_RT700_MIMXRT798S.IRTC_EnableClockOutputToPeripheral (*C function*), 524
drivers_RT_RT700_MIMXRT798S.IRTC_EnableInterrupts (*C function*), 517
drivers_RT_RT700_MIMXRT798S.IRTC_EnableSubsecondCounter (*C function*), 518
drivers_RT_RT700_MIMXRT798S.IRTC_GetAlarm (*C function*), 517
drivers_RT_RT700_MIMXRT798S.IRTC_GetDatetime (*C function*), 517
drivers_RT_RT700_MIMXRT798S.IRTC_GetDaylightTime (*C function*), 518
drivers_RT_RT700_MIMXRT798S.IRTC_GetDefaultConfig (*C function*), 517
drivers_RT_RT700_MIMXRT798S.IRTC_GetEnabledInterrupts (*C function*), 518
drivers_RT_RT700_MIMXRT798S.IRTC_GetStatusFlags (*C function*), 518
drivers_RT_RT700_MIMXRT798S.IRTC_GetSubsecondCount (*C function*), 519
drivers_RT_RT700_MIMXRT798S.IRTC_GetWakeupCount (*C function*), 519
drivers_RT_RT700_MIMXRT798S.IRTC_Init (*C function*), 516
drivers_RT_RT700_MIMXRT798S.irtc_interrupt_enable_t (*C type*), 522
drivers_RT_RT700_MIMXRT798S.IRTC_Reset (*C function*), 523
drivers_RT_RT700_MIMXRT798S.IRTC_SetAlarm (*C function*), 517
drivers_RT_RT700_MIMXRT798S.IRTC_SetWakeupCount (*C function*), 519
drivers_RT_RT700_MIMXRT798S.IRTC_SetWriteProtection (*C function*), 523
drivers_RT_RT700_MIMXRT798S.irtc_status_flags_t (*C type*), 522
drivers_RT_RT700_MIMXRT798S.IRTC_STATUS_W1C_BITS (*C macro*), 524
drivers_RT_RT700_MIMXRT798S.ITRC (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC0_DriverIRQHandler (*C function*), 528
drivers_RT_RT700_MIMXRT798S.ITRC_ClearAllStatus (*C function*), 527
drivers_RT_RT700_MIMXRT798S.ITRC_ClearStatus (*C function*), 526
drivers_RT_RT700_MIMXRT798S.ITRC_ClearStatus1 (*C function*), 526
drivers_RT_RT700_MIMXRT798S.ITRC_Deinit (*C function*), 527
drivers_RT_RT700_MIMXRT798S.itrc_enable_t (*C type*), 528
drivers_RT_RT700_MIMXRT798S.ITRC_GetStatus (*C function*), 526
drivers_RT_RT700_MIMXRT798S.ITRC_GetStatus1 (*C function*), 526
drivers_RT_RT700_MIMXRT798S.ITRC_Init (*C function*), 527
drivers_RT_RT700_MIMXRT798S.itrc_input_signals_t (*C type*), 528
drivers_RT_RT700_MIMXRT798S.itrc_lock_t (*C type*), 528
drivers_RT_RT700_MIMXRT798S.ITRC_OUT_COUNT (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.itrc_out_signals_t (*C type*), 528
drivers_RT_RT700_MIMXRT798S.ITRC_SetActionToEvent (*C function*), 525
drivers_RT_RT700_MIMXRT798S.ITRC_SetSWEvent0 (*C function*), 526
drivers_RT_RT700_MIMXRT798S.ITRC_SetSWEvent1 (*C function*), 526
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS1_IN17_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS1_IN19_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS1_IN24_21_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS1_IN32_25_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS1_IN36_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS1_IN46_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS_IN0_STATUS_MASK (*C macro*), 528
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS_IN1_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS_IN2_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS_IN3_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS_IN6_STATUS_MASK (*C macro*), 529

drivers_RT_RT700_MIMXRT798S.ITRC_STATUS_IN7_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS_IN8_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS_IN9_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.ITRC_STATUS_IN14_STATUS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.JPEG_ALIGN_SIZE (*C macro*), 538
drivers_RT_RT700_MIMXRT798S.JPEG_DECODER_Type (*C struct*), 538
drivers_RT_RT700_MIMXRT798S.jpegBufAddr (*C var*), 536
drivers_RT_RT700_MIMXRT798S.jpegBufSize (*C var*), 537
drivers_RT_RT700_MIMXRT798S.JPEGDEC_ClearStatusFlags (*C function*), 533
drivers_RT_RT700_MIMXRT798S.JPEGDEC_CLOCKS (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.jpegdec_config_t (*C type*), 532
drivers_RT_RT700_MIMXRT798S.JPEGDEC_ConfigDecoder (*C function*), 535
drivers_RT_RT700_MIMXRT798S.jpegdec_decoder_config_t (*C type*), 532
drivers_RT_RT700_MIMXRT798S.JPEGDEC_Deinit (*C function*), 532
drivers_RT_RT700_MIMXRT798S.jpegdec_descpt_t (*C type*), 532
drivers_RT_RT700_MIMXRT798S.JPEGDEC_DescptReset (*C function*), 536
drivers_RT_RT700_MIMXRT798S.JPEGDEC_DisableInterrupts (*C function*), 533
drivers_RT_RT700_MIMXRT798S.JPEGDEC_Enable (*C function*), 533
drivers_RT_RT700_MIMXRT798S.JPEGDEC_EnableInterrupts (*C function*), 533
drivers_RT_RT700_MIMXRT798S.JPEGDEC_EnableLinkedDescpt (*C function*), 536
drivers_RT_RT700_MIMXRT798S.JPEGDEC_EnableLp (*C function*), 533
drivers_RT_RT700_MIMXRT798S.JPEGDEC_EnableSlotNextDescpt (*C function*), 536
drivers_RT_RT700_MIMXRT798S.JPEGDEC_EnableSlots (*C function*), 534
drivers_RT_RT700_MIMXRT798S.jpegdec_endian_mode_t (*C type*), 532
drivers_RT_RT700_MIMXRT798S.JPEGDEC_GetActiveSlot (*C function*), 534
drivers_RT_RT700_MIMXRT798S.JPEGDEC_GetDefaultConfig (*C function*), 532
drivers_RT_RT700_MIMXRT798S.JPEGDEC_GetStatusFlags (*C function*), 533
drivers_RT_RT700_MIMXRT798S.JPEGDEC_Init (*C function*), 532
drivers_RT_RT700_MIMXRT798S.JPEGDEC_ParseHeader (*C function*), 535
drivers_RT_RT700_MIMXRT798S.jpegdec_pixel_depth_t (*C type*), 532
drivers_RT_RT700_MIMXRT798S.jpegdec_pixel_format_t (*C type*), 532
drivers_RT_RT700_MIMXRT798S.JPEGDEC_Reset (*C function*), 534
drivers_RT_RT700_MIMXRT798S.JPEGDEC_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.JPEGDEC_SetDecodeOption (*C function*), 535
drivers_RT_RT700_MIMXRT798S.JPEGDEC_SetEndian (*C function*), 534
drivers_RT_RT700_MIMXRT798S.JPEGDEC_SetJpegBuffer (*C function*), 534
drivers_RT_RT700_MIMXRT798S.JPEGDEC_SetOutputBuffer (*C function*), 534
drivers_RT_RT700_MIMXRT798S.JPEGDEC_SetSlotNextDescpt (*C function*), 535
drivers_RT_RT700_MIMXRT798S.JPEGDEC_StartDecode (*C function*), 534
drivers_RT_RT700_MIMXRT798S.kb_authenticate_t (*C struct*), 883
drivers_RT_RT700_MIMXRT798S.kb_load_sb_t (*C struct*), 883
drivers_RT_RT700_MIMXRT798S.kb_operation_t (*C enum*), 880
drivers_RT_RT700_MIMXRT798S.kb_operation_t.kRomAuthenticateImage (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S.kb_operation_t.kRomLoadImage (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S.kb_operation_t.kRomOperationCount (*C enumerator*), 880
drivers_RT_RT700_MIMXRT798S.kb_options_t (*C struct*), 883
drivers_RT_RT700_MIMXRT798S.kb_options_t.___unnamed94___ (*C union*), 884
drivers_RT_RT700_MIMXRT798S.kb_options_t.___unnamed94___.authenticate (*C var*), 884
drivers_RT_RT700_MIMXRT798S.kb_options_t.___unnamed94___.loadSB (*C var*), 884
drivers_RT_RT700_MIMXRT798S.kb_options_t.buffer (*C var*), 883
drivers_RT_RT700_MIMXRT798S.kb_options_t.version (*C var*), 883
drivers_RT_RT700_MIMXRT798S.kb_region_t (*C struct*), 883
drivers_RT_RT700_MIMXRT798S.keyScopeEnrolled (*C var*), 821
drivers_RT_RT700_MIMXRT798S.keyScopeStarted (*C var*), 821
drivers_RT_RT700_MIMXRT798S.kFlexcommIrqs (*C var*), 611
drivers_RT_RT700_MIMXRT798S.kGDET_0_9v (*C macro*), 429
drivers_RT_RT700_MIMXRT798S.kGDET_1_0v (*C macro*), 429
drivers_RT_RT700_MIMXRT798S.kGDET_1_1v (*C macro*), 429

drivers_RT_RT700_MIMXRT798S.kPUF_AcAuthFailedPhase1 (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_AcAuthFailedPhase2 (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_AcCorruptedPhase1 (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_AcCorruptedPhase2 (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_AcNotForThisProductPhase1 (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_AcNotForThisProductPhase2 (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_ContextIncorrect (*C macro*), 826
drivers_RT_RT700_MIMXRT798S.kPUF_DestinationNotAllowed (*C macro*), 826
drivers_RT_RT700_MIMXRT798S.kPUF_EndianBig (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_EndianLittle (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_Failure (*C macro*), 826
drivers_RT_RT700_MIMXRT798S.kPUF_KeyAllowAll (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_KeyAllowKeyBus (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_KeyAllowRegister (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_KeyDestInvalid (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_KeyDestKeyBus (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_KeyDestRegister (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_NonsecurePrivilege (*C macro*), 826
drivers_RT_RT700_MIMXRT798S.kPUF_NonsecureUser (*C macro*), 826
drivers_RT_RT700_MIMXRT798S.kPUF_QualityVerificationFail (*C macro*), 826
drivers_RT_RT700_MIMXRT798S.kPUF_ResultOK (*C macro*), 825
drivers_RT_RT700_MIMXRT798S.kPUF_SecurePrivilege (*C macro*), 826
drivers_RT_RT700_MIMXRT798S.kPUF_SecureUser (*C macro*), 826
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_AuthFail (*C macro*), 753
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_Fail (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_InvalidArgument (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_InvalidDataFormat (*C macro*), 753
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_IskCertSignatureOffsetTooSmall (*C macro*), 753
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_IskCertUserDataTooBig (*C macro*), 753
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_IvCounterOverflow (*C macro*), 753
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_KeyNotAvailable (*C macro*), 753
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_KeyNotLoaded (*C macro*), 753
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_MemcpyFail (*C macro*), 753
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_OperationNotAvaialable (*C macro*), 753
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_RequestTimeout (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_SelftestFail (*C macro*), 753
drivers_RT_RT700_MIMXRT798S.kStatus_NBOOT_Success (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.LCDIF_ADDR_CPU_2_IP (*C macro*), 577
drivers_RT_RT700_MIMXRT798S.LCDIF_ALIGN_ADDR (*C macro*), 577
drivers_RT_RT700_MIMXRT798S.LCDIF_CLOCKS (*C macro*), 258
drivers_RT_RT700_MIMXRT798S.lcdif_cursor_config_t (*C type*), 576
drivers_RT_RT700_MIMXRT798S.lcdif_cursor_format_t (*C type*), 576
drivers_RT_RT700_MIMXRT798S.LCDIF_CURSOR_SIZE (*C macro*), 577
drivers_RT_RT700_MIMXRT798S.LCDIF_CursorGetDefaultConfig (*C function*), 565
drivers_RT_RT700_MIMXRT798S.lcdif_dbi_config_t (*C type*), 576
drivers_RT_RT700_MIMXRT798S.lcdif_dbi_out_format_t (*C type*), 576
drivers_RT_RT700_MIMXRT798S.lcdif_dbi_out_swizzle_t (*C type*), 576
drivers_RT_RT700_MIMXRT798S.lcdif_dbi_type_t (*C type*), 576
drivers_RT_RT700_MIMXRT798S.LCDIF_DbiIsTypeCFifoFull (*C function*), 559
drivers_RT_RT700_MIMXRT798S.LCDIF_DbiModeGetDefaultConfig (*C function*), 558
drivers_RT_RT700_MIMXRT798S.LCDIF_DbiModeSetConfig (*C function*), 558
drivers_RT_RT700_MIMXRT798S.LCDIF_DbiReset (*C function*), 558
drivers_RT_RT700_MIMXRT798S.LCDIF_DbiSelectArea (*C function*), 559
drivers_RT_RT700_MIMXRT798S.LCDIF_DbiSendCommand (*C function*), 559
drivers_RT_RT700_MIMXRT798S.LCDIF_DbiSendCommandAndData (*C function*), 559
drivers_RT_RT700_MIMXRT798S.LCDIF_DbiSendData (*C function*), 559
drivers_RT_RT700_MIMXRT798S.LCDIF_DbiWriteMem (*C function*), 560

drivers_RT_RT700_MIMXRT798S.LCDIF_Deinit (*C function*), 557
drivers_RT_RT700_MIMXRT798S.LCDIF_DisableInterrupts (*C function*), 565
drivers_RT_RT700_MIMXRT798S.lcdif_dither_config_t (*C type*), 576
drivers_RT_RT700_MIMXRT798S.lcdif_dpi_config_t (*C type*), 575
drivers_RT_RT700_MIMXRT798S.LCDIF_DpiModeGetDefaultConfig (*C function*), 557
drivers_RT_RT700_MIMXRT798S.LCDIF_DpiModeSetConfig (*C function*), 557
drivers_RT_RT700_MIMXRT798S.LCDIF_EnableGamma (*C function*), 564
drivers_RT_RT700_MIMXRT798S.LCDIF_EnableInterrupts (*C function*), 565
drivers_RT_RT700_MIMXRT798S.LCDIF_EnablePanel (*C function*), 564
drivers_RT_RT700_MIMXRT798S.LCDIF_EnableUpdate (*C function*), 564
drivers_RT_RT700_MIMXRT798S.lcdif_endian_mode_t (*C type*), 576
drivers_RT_RT700_MIMXRT798S.LCDIF_FB_ALIGN (*C macro*), 577
drivers_RT_RT700_MIMXRT798S.lcdif_fb_config_t (*C type*), 575
drivers_RT_RT700_MIMXRT798S.lcdif_fb_format_t (*C type*), 575
drivers_RT_RT700_MIMXRT798S.LCDIF_FB_UV_ALIGN (*C macro*), 577
drivers_RT_RT700_MIMXRT798S.LCDIF_FRAMEBUFFERCONFIG0_OUTPUT_MASK (*C macro*), 577
drivers_RT_RT700_MIMXRT798S.LCDIF_FrameBufferGetDefaultConfig (*C function*), 560
drivers_RT_RT700_MIMXRT798S.LCDIF_GAMMA_INDEX_MAX (*C macro*), 577
drivers_RT_RT700_MIMXRT798S.LCDIF_GetAndClearInterruptPendingFlags (*C function*), 565
drivers_RT_RT700_MIMXRT798S.LCDIF_GetPorterDuffConfig (*C function*), 563
drivers_RT_RT700_MIMXRT798S.LCDIF_Init (*C function*), 557
drivers_RT_RT700_MIMXRT798S.lcdif_layer_alpha_blend_config_t (*C struct*), 578
drivers_RT_RT700_MIMXRT798S.lcdif_layer_alpha_blend_config_t.dstAlphaMode (*C var*), 579
drivers_RT_RT700_MIMXRT798S.lcdif_layer_alpha_blend_config_t.dstFactorMode (*C var*), 579
drivers_RT_RT700_MIMXRT798S.lcdif_layer_alpha_blend_config_t.dstGlobalAlpha (*C var*), 579
drivers_RT_RT700_MIMXRT798S.lcdif_layer_alpha_blend_config_t.dstGlobalAlphaMode (*C var*), 579
drivers_RT_RT700_MIMXRT798S.lcdif_layer_alpha_blend_config_t.enable (*C var*), 578
drivers_RT_RT700_MIMXRT798S.lcdif_layer_alpha_blend_config_t.srcAlphaMode (*C var*), 578
drivers_RT_RT700_MIMXRT798S.lcdif_layer_alpha_blend_config_t.srcFactorMode (*C var*), 578
drivers_RT_RT700_MIMXRT798S.lcdif_layer_alpha_blend_config_t.srcGlobalAlpha (*C var*), 579
drivers_RT_RT700_MIMXRT798S.lcdif_layer_alpha_blend_config_t.srcGlobalAlphaMode (*C var*), 578
drivers_RT_RT700_MIMXRT798S.lcdif_layer_alpha_blend_config_t.useDstAlpha (*C var*), 579
drivers_RT_RT700_MIMXRT798S.lcdif_layer_alpha_blend_config_t.useSrcAlpha (*C var*), 578
drivers_RT_RT700_MIMXRT798S.lcdif_layer_colorkey_t (*C type*), 575
drivers_RT_RT700_MIMXRT798S.lcdif_layer_convert_standard_t (*C type*), 575
drivers_RT_RT700_MIMXRT798S.lcdif_layer_decompress_mode_t (*C type*), 575
drivers_RT_RT700_MIMXRT798S.lcdif_layer_input_order_t (*C type*), 575
drivers_RT_RT700_MIMXRT798S.lcdif_layer_order_t (*C type*), 576
drivers_RT_RT700_MIMXRT798S.lcdif_layer_rotate_flip_t (*C type*), 575
drivers_RT_RT700_MIMXRT798S.LCDIF_MAKE_BACKGROUND_VALUE (*C macro*), 577
drivers_RT_RT700_MIMXRT798S.LCDIF_MAKE_CURSOR_COLOR (*C macro*), 577
drivers_RT_RT700_MIMXRT798S.LCDIF_MAKE_GAMMA_VALUE (*C macro*), 577
drivers_RT_RT700_MIMXRT798S.lcdif_output_format_t (*C type*), 575
drivers_RT_RT700_MIMXRT798S.lcdif_panel_config_t (*C type*), 576
drivers_RT_RT700_MIMXRT798S.LCDIF_PanelGetDefaultConfig (*C function*), 563
drivers_RT_RT700_MIMXRT798S.lcdif_porter_duff_blend_mode_t (*C type*), 575
drivers_RT_RT700_MIMXRT798S.LCDIF_Reset (*C function*), 564
drivers_RT_RT700_MIMXRT798S.LCDIF_RSTS (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.LCDIF_SetCursorBufferAddress (*C function*), 566
drivers_RT_RT700_MIMXRT798S.LCDIF_SetCursorColor (*C function*), 566
drivers_RT_RT700_MIMXRT798S.LCDIF_SetCursorConfig (*C function*), 565
drivers_RT_RT700_MIMXRT798S.LCDIF_SetCursorHotspotPosition (*C function*), 566
drivers_RT_RT700_MIMXRT798S.LCDIF_SetDitherConfig (*C function*), 564
drivers_RT_RT700_MIMXRT798S.LCDIF_SetEndianMode (*C function*), 563
drivers_RT_RT700_MIMXRT798S.LCDIF_SetFrameBufferAddr (*C function*), 560

drivers_RT_RT700_MIMXRT798S.LCDIF_SetFrameBufferBackground (*C function*), 563
drivers_RT_RT700_MIMXRT798S.LCDIF_SetFrameBufferConfig (*C function*), 560
drivers_RT_RT700_MIMXRT798S.LCDIF_SetFrameBufferPosition (*C function*), 561
drivers_RT_RT700_MIMXRT798S.LCDIF_SetFrameBufferStride (*C function*), 560
drivers_RT_RT700_MIMXRT798S.LCDIF_SetFrameBufferUVAddr (*C function*), 561
drivers_RT_RT700_MIMXRT798S.LCDIF_SetFrameBufferUVStride (*C function*), 561
drivers_RT_RT700_MIMXRT798S.LCDIF_SetGammaData (*C function*), 565
drivers_RT_RT700_MIMXRT798S.LCDIF_SetLayerOrder (*C function*), 563
drivers_RT_RT700_MIMXRT798S.LCDIF_SetOverlayLayerAddr (*C function*), 562
drivers_RT_RT700_MIMXRT798S.LCDIF_SetOverlayLayerConfig (*C function*), 561
drivers_RT_RT700_MIMXRT798S.LCDIF_SetOverlayLayerPosition (*C function*), 561
drivers_RT_RT700_MIMXRT798S.LCDIF_SetOverlayLayerStride (*C function*), 562
drivers_RT_RT700_MIMXRT798S.LCDIF_SetOverlayLayerUVAddr (*C function*), 562
drivers_RT_RT700_MIMXRT798S.LCDIF_SetOverlayLayerUVStride (*C function*), 563
drivers_RT_RT700_MIMXRT798S.LCDIF_SetPanelConfig (*C function*), 563
drivers_RT_RT700_MIMXRT798S.LCDIF_SetUpdateReady (*C function*), 564
drivers_RT_RT700_MIMXRT798S.LCDIF_Start (*C function*), 564
drivers_RT_RT700_MIMXRT798S.LCDIFPIXELCLKDIV_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.LCDIFPIXELCLKSEL_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.LDO (*C var*), 818
drivers_RT_RT700_MIMXRT798S.ldr_buf_t (*C type*), 880
drivers_RT_RT700_MIMXRT798S.ldr_Context_v3_t (*C type*), 880
drivers_RT_RT700_MIMXRT798S.load_keyblob (*C struct*), 882
drivers_RT_RT700_MIMXRT798S.LOWFREQCLKDIV_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.lowPowerMode (*C var*), 267
drivers_RT_RT700_MIMXRT798S.lp_bit_t (*C type*), 806
drivers_RT_RT700_MIMXRT798S.lp_bits (*C enum*), 800
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_ACMP0_STOP (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_ADC0_STOP (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_EDMA2_STOP (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_EDMA3_STOP (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_EZHV_EXIT_WAIT (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_EZHV_HALT (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_EZHV_STOP (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_FC17_B_LPREQ (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_FC17_F_LPREQ (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_FC18_B_LPREQ (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_FC18_F_LPREQ (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_FC19_B_LPREQ (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_FC19_F_LPREQ (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_FC20_B_LPREQ (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_FC20_F_LPREQ (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_FLEXIO_B_LPREQ (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_FLEXIO_F_LPREQ (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_GDET3_LPREQ (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_GPIO8_LPREQ (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_GPIO9_LPREQ (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_GPIO10_LPREQ (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_I3C2_STOP (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_I3C3_STOP (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_LPI2C15_B_LPREQ (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_LPI2C15_F_LPREQ (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_LPSPI14_B_LPREQ (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_LPSPI14_F_LPREQ (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_LPSPI16_B_LPREQ (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_LPSPI16_F_LPREQ (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_MICFIL_STOP (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_NIC0_LPREQ (*C enumerator*), 801

drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_NIC1_LPREQ (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_RTC_STOP (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_SAI3_LPREQ (*C enumerator*), 802
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_VGPU_LPREQ (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.lp_bits.kPower_XSPI2_STOP (*C enumerator*), 801
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_Deinit (*C function*), 611
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_GetBaseAddress (*C function*), 611
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_GetInstance (*C function*), 611
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_GetInterruptStatus (*C function*), 611
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_Init (*C function*), 611
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_PERIPH_T (*C enum*), 610
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_PERIPH_T.LP_FLEXCOMM_PERIPH_LPI2C (*C enumerator*), 611
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_PERIPH_T.LP_FLEXCOMM_PERIPH_LPI2CAndLPUART (*C enumerator*), 611
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_PERIPH_T.LP_FLEXCOMM_PERIPH_LPSPI (*C enumerator*), 611
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_PERIPH_T.LP_FLEXCOMM_PERIPH_LPUART (*C enumerator*), 611
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_PERIPH_T.LP_FLEXCOMM_PERIPH_NONE (*C enumerator*), 610
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_RSTS (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.LP_FLEXCOMM_SetIRQHandler (*C function*), 611
drivers_RT_RT700_MIMXRT798S.LPADC_CALIBRATION_READY_TIMEOUT (*C macro*), 592
drivers_RT_RT700_MIMXRT798S.lpadc_calibration_value_t (*C type*), 592
drivers_RT_RT700_MIMXRT798S.LPADC_ClearStatusFlags (*C function*), 594
drivers_RT_RT700_MIMXRT798S.LPADC_ClearTriggerStatusFlags (*C function*), 594
drivers_RT_RT700_MIMXRT798S.LPADC_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.lpadc_config_t (*C struct*), 601
drivers_RT_RT700_MIMXRT798S.lpadc_config_t.conversionAverageMode (*C var*), 601
drivers_RT_RT700_MIMXRT798S.lpadc_config_t.convPauseDelay (*C var*), 602
drivers_RT_RT700_MIMXRT798S.lpadc_config_t.enableAnalogPreliminary (*C var*), 602
drivers_RT_RT700_MIMXRT798S.lpadc_config_t.enableConvPause (*C var*), 602
drivers_RT_RT700_MIMXRT798S.lpadc_config_t.enableInDozeMode (*C var*), 601
drivers_RT_RT700_MIMXRT798S.lpadc_config_t.enableInternalClock (*C var*), 601
drivers_RT_RT700_MIMXRT798S.lpadc_config_t.enableVref1LowVoltage (*C var*), 601
drivers_RT_RT700_MIMXRT798S.lpadc_config_t.FIFOWatermark (*C var*), 602
drivers_RT_RT700_MIMXRT798S.lpadc_config_t.powerLevelMode (*C var*), 602
drivers_RT_RT700_MIMXRT798S.lpadc_config_t.powerUpDelay (*C var*), 602
drivers_RT_RT700_MIMXRT798S.lpadc_config_t.referenceVoltageSource (*C var*), 602
drivers_RT_RT700_MIMXRT798S.lpadc_config_t.triggerPriorityPolicy (*C var*), 602
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t (*C struct*), 602
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.chainedNextCommandNumber (*C var*), 603
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.channelBNumber (*C var*), 603
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.channelBScaleMode (*C var*), 602
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.channelNumber (*C var*), 603
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.conversionResolutionMode (*C var*), 603
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.enableAutoChannelIncrement (*C var*), 603
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.enableWaitTrigger (*C var*), 603
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.hardwareAverageMode (*C var*), 603
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.hardwareCompareMode (*C var*), 603

drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.hardwareCompareValueHigh (*C var*), 603
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.hardwareCompareValueLow (*C var*), 603
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.loopCount (*C var*), 603
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.sampleChannelMode (*C var*), 602
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.sampleScaleMode (*C var*), 602
drivers_RT_RT700_MIMXRT798S.lpadc_conv_command_config_t.sampleTimeMode (*C var*), 603
drivers_RT_RT700_MIMXRT798S.lpadc_conv_result_t (*C struct*), 604
drivers_RT_RT700_MIMXRT798S.lpadc_conv_result_t.commandIdSource (*C var*), 604
drivers_RT_RT700_MIMXRT798S.lpadc_conv_result_t.convValue (*C var*), 604
drivers_RT_RT700_MIMXRT798S.lpadc_conv_result_t.loopCountIndex (*C var*), 604
drivers_RT_RT700_MIMXRT798S.lpadc_conv_result_t.triggerIdSource (*C var*), 604
drivers_RT_RT700_MIMXRT798S.lpadc_conv_trigger_config_t (*C struct*), 603
drivers_RT_RT700_MIMXRT798S.lpadc_conv_trigger_config_t.delayPower (*C var*), 604
drivers_RT_RT700_MIMXRT798S.lpadc_conv_trigger_config_t.enableHardwareTrigger (*C var*), 604
drivers_RT_RT700_MIMXRT798S.lpadc_conv_trigger_config_t.priority (*C var*), 604
drivers_RT_RT700_MIMXRT798S.lpadc_conv_trigger_config_t.targetCommandId (*C var*), 604
drivers_RT_RT700_MIMXRT798S.lpadc_conversion_average_mode_t (*C type*), 591
drivers_RT_RT700_MIMXRT798S.LPADC_CONVERSION_COMPLETE_TIMEOUT (*C macro*), 592
drivers_RT_RT700_MIMXRT798S.lpadc_conversion_resolution_mode_t (*C type*), 591
drivers_RT_RT700_MIMXRT798S.LPADC_Deinit (*C function*), 593
drivers_RT_RT700_MIMXRT798S.LPADC_DisableInterrupts (*C function*), 594
drivers_RT_RT700_MIMXRT798S.LPADC_DoAutoCalibration (*C function*), 597
drivers_RT_RT700_MIMXRT798S.LPADC_DoOffsetCalibration (*C function*), 598
drivers_RT_RT700_MIMXRT798S.LPADC_DoResetConfig (*C function*), 593
drivers_RT_RT700_MIMXRT798S.LPADC_DoResetFIFO (*C function*), 593
drivers_RT_RT700_MIMXRT798S.LPADC_DoSoftwareTrigger (*C function*), 596
drivers_RT_RT700_MIMXRT798S.LPADC_Enable (*C function*), 593
drivers_RT_RT700_MIMXRT798S.LPADC_EnableCalibration (*C function*), 597
drivers_RT_RT700_MIMXRT798S.LPADC_EnableExtraCycle (*C function*), 600
drivers_RT_RT700_MIMXRT798S.LPADC_EnableFIFOWatermarkDMA (*C function*), 595
drivers_RT_RT700_MIMXRT798S.LPADC_EnableHardwareTriggerCommandSelection (*C function*), 596
drivers_RT_RT700_MIMXRT798S.LPADC_EnableHighSpeedConversionMode (*C function*), 600
drivers_RT_RT700_MIMXRT798S.LPADC_EnableInterrupts (*C function*), 594
drivers_RT_RT700_MIMXRT798S.LPADC_EnableJustifiedLeft (*C function*), 601
drivers_RT_RT700_MIMXRT798S.LPADC_EnableOffsetCalibration (*C function*), 598
drivers_RT_RT700_MIMXRT798S.LPADC_FinishAutoCalibration (*C function*), 599
drivers_RT_RT700_MIMXRT798S.LPADC_GAIN_CAL_READY_TIMEOUT (*C macro*), 592
drivers_RT_RT700_MIMXRT798S.LPADC_GET_ACTIVE_COMMAND_STATUS (*C macro*), 592
drivers_RT_RT700_MIMXRT798S.LPADC_GET_ACTIVE_TRIGGER_STATUE (*C macro*), 592
drivers_RT_RT700_MIMXRT798S.LPADC_GetCalibrationValue (*C function*), 599
drivers_RT_RT700_MIMXRT798S.LPADC_GetConvResult (*C function*), 595
drivers_RT_RT700_MIMXRT798S.LPADC_GetConvResultBlocking (*C function*), 595
drivers_RT_RT700_MIMXRT798S.LPADC_GetConvResultCount (*C function*), 595
drivers_RT_RT700_MIMXRT798S.LPADC_GetDefaultConfig (*C function*), 593
drivers_RT_RT700_MIMXRT798S.LPADC_GetDefaultConvCommandConfig (*C function*), 596
drivers_RT_RT700_MIMXRT798S.LPADC_GetDefaultConvTriggerConfig (*C function*), 595
drivers_RT_RT700_MIMXRT798S.LPADC_GetHighSpeedTrimValue (*C function*), 600
drivers_RT_RT700_MIMXRT798S.LPADC_GetOffsetValue (*C function*), 598
drivers_RT_RT700_MIMXRT798S.LPADC_GetStatusFlags (*C function*), 593
drivers_RT_RT700_MIMXRT798S.LPADC_GetTriggerStatusFlags (*C function*), 594
drivers_RT_RT700_MIMXRT798S.LPADC_GetTuneValue (*C function*), 601
drivers_RT_RT700_MIMXRT798S.lpadc_hardware_average_mode_t (*C type*), 590
drivers_RT_RT700_MIMXRT798S.lpadc_hardware_compare_mode_t (*C type*), 591

drivers_RT_RT700_MIMXRT798S.LPADC_Init (*C function*), 593
drivers_RT_RT700_MIMXRT798S.lpadc_offset_calibration_mode_t (*C type*), 591
drivers_RT_RT700_MIMXRT798S.lpadc_power_level_mode_t (*C type*), 591
drivers_RT_RT700_MIMXRT798S.LPADC_PrepareAutoCalibration (*C function*), 599
drivers_RT_RT700_MIMXRT798S.lpadc_reference_voltage_source_t (*C type*), 591
drivers_RT_RT700_MIMXRT798S.LPADC_RequestHighSpeedModeTrim (*C function*), 600
drivers_RT_RT700_MIMXRT798S.lpadc_sample_channel_mode_t (*C type*), 590
drivers_RT_RT700_MIMXRT798S.lpadc_sample_scale_mode_t (*C type*), 590
drivers_RT_RT700_MIMXRT798S.lpadc_sample_time_mode_t (*C type*), 591
drivers_RT_RT700_MIMXRT798S.LPADC_SetCalibrationValue (*C function*), 599
drivers_RT_RT700_MIMXRT798S.LPADC_SetConvCommandConfig (*C function*), 596
drivers_RT_RT700_MIMXRT798S.LPADC_SetConvTriggerConfig (*C function*), 595
drivers_RT_RT700_MIMXRT798S.LPADC_SetHighSpeedTrimValue (*C function*), 600
drivers_RT_RT700_MIMXRT798S.LPADC_SetOffsetCalibrationMode (*C function*), 598
drivers_RT_RT700_MIMXRT798S.LPADC_SetOffsetValue (*C function*), 597, 598
drivers_RT_RT700_MIMXRT798S.LPADC_SetTuneValue (*C function*), 600
drivers_RT_RT700_MIMXRT798S.lpadc_trigger_priority_policy_t (*C type*), 591
drivers_RT_RT700_MIMXRT798S.lpadc_tune_value_t (*C type*), 592
drivers_RT_RT700_MIMXRT798S.LPADC_USE_FIXED_POINT_GAIN_CALCULATION (*C macro*), 592
drivers_RT_RT700_MIMXRT798S.LPC_IOPCTL_DRIVER_VERSION (*C macro*), 514
drivers_RT_RT700_MIMXRT798S.lpflexcomm_irq_handler_t (*C type*), 611
drivers_RT_RT700_MIMXRT798S.LPI2C_CheckForBusyBus (*C function*), 614
drivers_RT_RT700_MIMXRT798S.LPI2C_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.lpi2c_data_match_config_mode_t (*C type*), 624
drivers_RT_RT700_MIMXRT798S.lpi2c_data_match_config_t (*C type*), 624
drivers_RT_RT700_MIMXRT798S.lpi2c_direction_t (*C type*), 624
drivers_RT_RT700_MIMXRT798S.lpi2c_host_request_polarity_t (*C type*), 624
drivers_RT_RT700_MIMXRT798S.lpi2c_host_request_source_t (*C type*), 624
drivers_RT_RT700_MIMXRT798S.lpi2c_master_config_t (*C type*), 624
drivers_RT_RT700_MIMXRT798S.lpi2c_master_edma_handle_t (*C type*), 629
drivers_RT_RT700_MIMXRT798S.lpi2c_master_edma_transfer_callback_t (*C type*), 629
drivers_RT_RT700_MIMXRT798S.lpi2c_master_handle_t (*C type*), 624
drivers_RT_RT700_MIMXRT798S.lpi2c_master_isr_t (*C type*), 625
drivers_RT_RT700_MIMXRT798S.lpi2c_master_pin_config_t (*C type*), 624
drivers_RT_RT700_MIMXRT798S.lpi2c_master_transfer_callback_t (*C type*), 624
drivers_RT_RT700_MIMXRT798S.lpi2c_master_transfer_t (*C type*), 624
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterCheckAndClearError (*C function*), 613
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterClearStatusFlags (*C function*), 614
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterConfigureDataMatch (*C function*), 613
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterCreateEDMAHandle (*C function*), 628
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterDeinit (*C function*), 613
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterDisableInterrupts (*C function*), 615
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterEnable (*C function*), 614
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterEnableDMA (*C function*), 615
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterEnableInterrupts (*C function*), 615
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterGetBusIdleState (*C function*), 616
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterGetDefaultConfig (*C function*), 613
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterGetEnabledInterrupts (*C function*), 615
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterGetFifoCounts (*C function*), 616
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterGetRxFifoAddress (*C function*), 615
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterGetStatusFlags (*C function*), 614
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterGetTxFifoAddress (*C function*), 615
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterInit (*C function*), 613
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterReceive (*C function*), 618
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterRepeatedStart (*C function*), 617
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterReset (*C function*), 614
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterSend (*C function*), 617

drivers_RT_RT700_MIMXRT798S.LPI2C_MasterSetBaudRate (*C function*), 616
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterSetWatermarks (*C function*), 616
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterStart (*C function*), 617
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterStop (*C function*), 618
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterTransferAbort (*C function*), 620
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterTransferAbortEDMA (*C function*), 629
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterTransferBlocking (*C function*), 619
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterTransferCreateHandle (*C function*), 619
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterTransferEDMA (*C function*), 628
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterTransferGetCount (*C function*), 620
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterTransferGetCountEDMA (*C function*), 628
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterTransferHandleIRQ (*C function*), 620
drivers_RT_RT700_MIMXRT798S.LPI2C_MasterTransferNonBlocking (*C function*), 619
drivers_RT_RT700_MIMXRT798S.lpi2c_slave_address_match_t (*C type*), 638
drivers_RT_RT700_MIMXRT798S.lpi2c_slave_config_t (*C type*), 638
drivers_RT_RT700_MIMXRT798S.lpi2c_slave_handle_t (*C type*), 639
drivers_RT_RT700_MIMXRT798S.lpi2c_slave_transfer_callback_t (*C type*), 639
drivers_RT_RT700_MIMXRT798S.lpi2c_slave_transfer_event_t (*C type*), 638
drivers_RT_RT700_MIMXRT798S.lpi2c_slave_transfer_t (*C type*), 639
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveClearStatusFlags (*C function*), 632
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveDeinit (*C function*), 631
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveDisableInterrupts (*C function*), 632
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveEnable (*C function*), 631
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveEnableAckStall (*C function*), 633
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveEnableDMA (*C function*), 633
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveEnableInterrupts (*C function*), 632
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveGetBusIdleState (*C function*), 633
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveGetDefaultConfig (*C function*), 630
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveGetEnabledInterrupts (*C function*), 633
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveGetReceivedAddress (*C function*), 634
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveGetStatusFlags (*C function*), 632
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveInit (*C function*), 631
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveReceive (*C function*), 634
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveReset (*C function*), 631
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveSend (*C function*), 634
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveTransferAbort (*C function*), 635
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveTransferCreateHandle (*C function*), 634
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveTransferGetCount (*C function*), 635
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveTransferHandleIRQ (*C function*), 636
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveTransferNonBlocking (*C function*), 635
drivers_RT_RT700_MIMXRT798S.LPI2C_SlaveTransmitAck (*C function*), 633
drivers_RT_RT700_MIMXRT798S.LPI2CFCLKDIV_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.LPI2CFCLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.LPSPI14CLKDIV_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.LPSPI14CLKSEL_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.LPSPI16CLKDIV_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.LPSPI16CLKSEL_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.LPSPI_CheckTransferArgument (*C function*), 645
drivers_RT_RT700_MIMXRT798S.LPSPI_ClearStatusFlags (*C function*), 644
drivers_RT_RT700_MIMXRT798S.lpspi_clock_phase_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.lpspi_clock_polarity_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.LPSPI_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.lpspi_data_out_config_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.LPSPI_Deinit (*C function*), 643
drivers_RT_RT700_MIMXRT798S.lpspi_delay_type_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.LPSPI_DisableDMA (*C function*), 645
drivers_RT_RT700_MIMXRT798S.LPSPI_DisableInterrupts (*C function*), 644
drivers_RT_RT700_MIMXRT798S.LPSPI_DUMMY_DATA (*C macro*), 661

drivers_RT_RT700_MIMXRT798S.LPSPI_Enable (*C function*), 643
drivers_RT_RT700_MIMXRT798S.LPSPI_EnableDMA (*C function*), 645
drivers_RT_RT700_MIMXRT798S.LPSPI_EnableInterrupts (*C function*), 644
drivers_RT_RT700_MIMXRT798S.LPSPI_FlushFifo (*C function*), 646
drivers_RT_RT700_MIMXRT798S.LPSPI_GetInstance (*C function*), 643
drivers_RT_RT700_MIMXRT798S.LPSPI_GetRxFifoCount (*C function*), 644
drivers_RT_RT700_MIMXRT798S.LPSPI_GetRxFifoSize (*C function*), 643
drivers_RT_RT700_MIMXRT798S.LPSPI_GetRxRegisterAddress (*C function*), 645
drivers_RT_RT700_MIMXRT798S.LPSPI_GetStatusFlags (*C function*), 643
drivers_RT_RT700_MIMXRT798S.LPSPI_GetTcr (*C function*), 644
drivers_RT_RT700_MIMXRT798S.LPSPI_GetTxFifoCount (*C function*), 644
drivers_RT_RT700_MIMXRT798S.LPSPI_GetTxFifoSize (*C function*), 643
drivers_RT_RT700_MIMXRT798S.LPSPI_GetTxRegisterAddress (*C function*), 645
drivers_RT_RT700_MIMXRT798S.lpspi_host_request_select_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.LPSPI_IsMaster (*C function*), 646
drivers_RT_RT700_MIMXRT798S.lpspi_master_config_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.lpspi_master_edma_handle_t (*C type*), 666
drivers_RT_RT700_MIMXRT798S.lpspi_master_edma_transfer_callback_t (*C type*), 666
drivers_RT_RT700_MIMXRT798S.lpspi_master_handle_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.LPSPI_MASTER_PCS_MASK (*C macro*), 661
drivers_RT_RT700_MIMXRT798S.LPSPI_MASTER_PCS_SHIFT (*C macro*), 661
drivers_RT_RT700_MIMXRT798S.lpspi_master_slave_mode_t (*C type*), 659
drivers_RT_RT700_MIMXRT798S.lpspi_master_transfer_callback_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.LPSPI_MASTER_WIDTH_MASK (*C macro*), 661
drivers_RT_RT700_MIMXRT798S.LPSPI_MASTER_WIDTH_SHIFT (*C macro*), 661
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterGetDefaultConfig (*C function*), 642
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterInit (*C function*), 642
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterSetBaudRate (*C function*), 647
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterSetDelayScaler (*C function*), 648
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterSetDelayTimes (*C function*), 648
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterTransferAbort (*C function*), 650
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterTransferAbortEDMA (*C function*), 669
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterTransferBlocking (*C function*), 649
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterTransferCreateHandle (*C function*), 649
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterTransferCreateHandleEDMA (*C function*), 667
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterTransferEDMA (*C function*), 667
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterTransferEDMALite (*C function*), 668
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterTransferGetCount (*C function*), 650
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterTransferGetCountEDMA (*C function*), 669
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterTransferHandleIRQ (*C function*), 651
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterTransferNonBlocking (*C function*), 650
drivers_RT_RT700_MIMXRT798S.LPSPI_MasterTransferPrepareEDMALite (*C function*), 668
drivers_RT_RT700_MIMXRT798S.lpspi_match_config_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.lpspi_pcs_function_config_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.lpspi_pcs_polarity_config_t (*C type*), 659
drivers_RT_RT700_MIMXRT798S.lpspi_pin_config_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.LPSPI_ReadData (*C function*), 649
drivers_RT_RT700_MIMXRT798S.LPSPI_Reset (*C function*), 643
drivers_RT_RT700_MIMXRT798S.LPSPI_SelectTransferPCS (*C function*), 646
drivers_RT_RT700_MIMXRT798S.LPSPI_SetAllPcsPolarity (*C function*), 647
drivers_RT_RT700_MIMXRT798S.LPSPI_SetDummyData (*C function*), 649
drivers_RT_RT700_MIMXRT798S.LPSPI_SetFifoWatermarks (*C function*), 646
drivers_RT_RT700_MIMXRT798S.LPSPI_SetFrameSize (*C function*), 647
drivers_RT_RT700_MIMXRT798S.LPSPI_SetMasterSlaveMode (*C function*), 646
drivers_RT_RT700_MIMXRT798S.LPSPI_SetPCSContinous (*C function*), 646
drivers_RT_RT700_MIMXRT798S.lpspi_shift_direction_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.lpspi_slave_config_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.lpspi_slave_edma_handle_t (*C type*), 666

drivers_RT_RT700_MIMXRT798S.lpspi_slave_edma_transfer_callback_t (*C type*), 667
drivers_RT_RT700_MIMXRT798S.lpspi_slave_handle_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.LPSPI_SLAVE_PCS_MASK (*C macro*), 661
drivers_RT_RT700_MIMXRT798S.LPSPI_SLAVE_PCS_SHIFT (*C macro*), 661
drivers_RT_RT700_MIMXRT798S.lpspi_slave_transfer_callback_t (*C type*), 661
drivers_RT_RT700_MIMXRT798S.LPSPI_SlaveGetDefaultConfig (*C function*), 642
drivers_RT_RT700_MIMXRT798S.LPSPI_SlaveInit (*C function*), 642
drivers_RT_RT700_MIMXRT798S.LPSPI_SlaveTransferAbort (*C function*), 652
drivers_RT_RT700_MIMXRT798S.LPSPI_SlaveTransferAbortEDMA (*C function*), 670
drivers_RT_RT700_MIMXRT798S.LPSPI_SlaveTransferCreateHandle (*C function*), 651
drivers_RT_RT700_MIMXRT798S.LPSPI_SlaveTransferCreateHandleEDMA (*C function*), 669
drivers_RT_RT700_MIMXRT798S.LPSPI_SlaveTransferEDMA (*C function*), 670
drivers_RT_RT700_MIMXRT798S.LPSPI_SlaveTransferGetCount (*C function*), 651
drivers_RT_RT700_MIMXRT798S.LPSPI_SlaveTransferGetCountEDMA (*C function*), 670
drivers_RT_RT700_MIMXRT798S.LPSPI_SlaveTransferHandleIRQ (*C function*), 652
drivers_RT_RT700_MIMXRT798S.LPSPI_SlaveTransferNonBlocking (*C function*), 651
drivers_RT_RT700_MIMXRT798S.lpspi_transfer_t (*C type*), 661
drivers_RT_RT700_MIMXRT798S.lpspi_transfer_width_t (*C type*), 660
drivers_RT_RT700_MIMXRT798S.lpspi_which_pcs_t (*C type*), 659
drivers_RT_RT700_MIMXRT798S.LPSPI_WriteData (*C function*), 649
drivers_RT_RT700_MIMXRT798S.LPUART_ClearStatusFlags (*C function*), 675
drivers_RT_RT700_MIMXRT798S.LPUART_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.lpuart_config_t (*C type*), 691
drivers_RT_RT700_MIMXRT798S.lpuart_data_bits_t (*C type*), 690
drivers_RT_RT700_MIMXRT798S.LPUART_Deinit (*C function*), 673
drivers_RT_RT700_MIMXRT798S.LPUART_DisableInterrupts (*C function*), 676
drivers_RT_RT700_MIMXRT798S.lpuart_edma_handle_t (*C type*), 698
drivers_RT_RT700_MIMXRT798S.lpuart_edma_transfer_callback_t (*C type*), 698
drivers_RT_RT700_MIMXRT798S.LPUART_Enable9bitMode (*C function*), 674
drivers_RT_RT700_MIMXRT798S.LPUART_EnableInterrupts (*C function*), 676
drivers_RT_RT700_MIMXRT798S.LPUART_EnableMatchAddress (*C function*), 674
drivers_RT_RT700_MIMXRT798S.LPUART_EnableRx (*C function*), 677
drivers_RT_RT700_MIMXRT798S.LPUART_EnableRxDMA (*C function*), 677
drivers_RT_RT700_MIMXRT798S.LPUART_EnableTx (*C function*), 677
drivers_RT_RT700_MIMXRT798S.LPUART_EnableTxDMA (*C function*), 677
drivers_RT_RT700_MIMXRT798S.LPUART_GetDataRegisterAddress (*C function*), 677
drivers_RT_RT700_MIMXRT798S.LPUART_GetDefaultConfig (*C function*), 673
drivers_RT_RT700_MIMXRT798S.LPUART_GetEnabledInterrupts (*C function*), 676
drivers_RT_RT700_MIMXRT798S.LPUART_GetInstance (*C function*), 677
drivers_RT_RT700_MIMXRT798S.LPUART_GetRxFifoCount (*C function*), 678
drivers_RT_RT700_MIMXRT798S.LPUART_GetStatusFlags (*C function*), 675
drivers_RT_RT700_MIMXRT798S.LPUART_GetTxFifoCount (*C function*), 678
drivers_RT_RT700_MIMXRT798S.lpuart_handle_t (*C type*), 691
drivers_RT_RT700_MIMXRT798S.lpuart_idle_config_t (*C type*), 690
drivers_RT_RT700_MIMXRT798S.lpuart_idle_type_select_t (*C type*), 690
drivers_RT_RT700_MIMXRT798S.LPUART_Init (*C function*), 673
drivers_RT_RT700_MIMXRT798S.lpuart_irq_handler_t (*C type*), 691
drivers_RT_RT700_MIMXRT798S.lpuart_parity_mode_t (*C type*), 690
drivers_RT_RT700_MIMXRT798S.LPUART_ReadBlocking (*C function*), 679
drivers_RT_RT700_MIMXRT798S.LPUART_ReadBlocking16bit (*C function*), 679
drivers_RT_RT700_MIMXRT798S.LPUART_ReadByte (*C function*), 678
drivers_RT_RT700_MIMXRT798S.LPUART_ReceiveEDMA (*C function*), 697
drivers_RT_RT700_MIMXRT798S.LPUART_SendAddress (*C function*), 678
drivers_RT_RT700_MIMXRT798S.LPUART_SendEDMA (*C function*), 696
drivers_RT_RT700_MIMXRT798S.LPUART_SetBaudRate (*C function*), 673
drivers_RT_RT700_MIMXRT798S.LPUART_SetMatchAddress (*C function*), 674
drivers_RT_RT700_MIMXRT798S.LPUART_SetRxFifoWatermark (*C function*), 675
drivers_RT_RT700_MIMXRT798S.LPUART_SetTxFifoWatermark (*C function*), 675

drivers_RT_RT700_MIMXRT798S.LPUART_SoftwareReset (*C function*), 672
drivers_RT_RT700_MIMXRT798S.lpuart_stop_bit_count_t (*C type*), 690
drivers_RT_RT700_MIMXRT798S.lpuart_timeout_condition_t (*C type*), 690
drivers_RT_RT700_MIMXRT798S.lpuart_timeout_config_t (*C type*), 690
drivers_RT_RT700_MIMXRT798S.lpuart_timeout_counter_config_t (*C type*), 690
drivers_RT_RT700_MIMXRT798S.lpuart_transfer_callback_t (*C type*), 691
drivers_RT_RT700_MIMXRT798S.lpuart_transfer_t (*C type*), 691
drivers_RT_RT700_MIMXRT798S.LPUART_TransferAbortReceive (*C function*), 682
drivers_RT_RT700_MIMXRT798S.LPUART_TransferAbortReceiveEDMA (*C function*), 697
drivers_RT_RT700_MIMXRT798S.LPUART_TransferAbortSend (*C function*), 681
drivers_RT_RT700_MIMXRT798S.LPUART_TransferAbortSendEDMA (*C function*), 697
drivers_RT_RT700_MIMXRT798S.LPUART_TransferCreateHandle (*C function*), 680
drivers_RT_RT700_MIMXRT798S.LPUART_TransferCreateHandleEDMA (*C function*), 695
drivers_RT_RT700_MIMXRT798S.LPUART_TransferCreateHandleEDMAExt (*C function*), 696
drivers_RT_RT700_MIMXRT798S.LPUART_TransferEdmaHandleIRQ (*C function*), 698
drivers_RT_RT700_MIMXRT798S.LPUART_TransferEnable16Bit (*C function*), 675
drivers_RT_RT700_MIMXRT798S.LPUART_TransferGetReceiveCount (*C function*), 683
drivers_RT_RT700_MIMXRT798S.LPUART_TransferGetReceiveCountEDMA (*C function*), 697
drivers_RT_RT700_MIMXRT798S.LPUART_TransferGetRxRingBufferLength (*C function*), 681
drivers_RT_RT700_MIMXRT798S.LPUART_TransferGetSendCount (*C function*), 682
drivers_RT_RT700_MIMXRT798S.LPUART_TransferGetSendCountEDMA (*C function*), 697
drivers_RT_RT700_MIMXRT798S.LPUART_TransferHandleErrorIRQ (*C function*), 683
drivers_RT_RT700_MIMXRT798S.LPUART_TransferHandleIRQ (*C function*), 683
drivers_RT_RT700_MIMXRT798S.LPUART_TransferReceiveNonBlocking (*C function*), 682
drivers_RT_RT700_MIMXRT798S.LPUART_TransferSendNonBlocking (*C function*), 680
drivers_RT_RT700_MIMXRT798S.LPUART_TransferStartRingBuffer (*C function*), 681
drivers_RT_RT700_MIMXRT798S.LPUART_TransferStopRingBuffer (*C function*), 681
drivers_RT_RT700_MIMXRT798S.lpuart_transmit_cts_config_t (*C type*), 690
drivers_RT_RT700_MIMXRT798S.lpuart_transmit_cts_source_t (*C type*), 690
drivers_RT_RT700_MIMXRT798S.lpuart_tx_callback_mode_t (*C type*), 698
drivers_RT_RT700_MIMXRT798S.LPUART_WriteBlocking (*C function*), 678
drivers_RT_RT700_MIMXRT798S.LPUART_WriteBlocking16bit (*C function*), 679
drivers_RT_RT700_MIMXRT798S.LPUART_WriteByte (*C function*), 678
drivers_RT_RT700_MIMXRT798S.lvl (*C var*), 818
drivers_RT_RT700_MIMXRT798S.lvl0 (*C var*), 818
drivers_RT_RT700_MIMXRT798S.lvl1 (*C var*), 818
drivers_RT_RT700_MIMXRT798S.lvl2 (*C var*), 818
drivers_RT_RT700_MIMXRT798S.lvl3 (*C var*), 818
drivers_RT_RT700_MIMXRT798S.main_pll_mult (*C var*), 266
drivers_RT_RT700_MIMXRT798S.main_pll_mult_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.main_pll_src (*C var*), 266
drivers_RT_RT700_MIMXRT798S.main_pll_src_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.MAINPLL0CLKSEL_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.MAINPLL0LOCKTIMEDIV2_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.MAKE_PD_BITS (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.MAKE_STATUS (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.MAKE_VERSION (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.MAX (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.MCUX_CS (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.MCUX_DEPRECATED (*C macro*), 542
drivers_RT_RT700_MIMXRT798S.MCUX_DEPRECATED_MACRO (*C macro*), 542
drivers_RT_RT700_MIMXRT798S.MCUX_EXPERIMENTAL (*C macro*), 542
drivers_RT_RT700_MIMXRT798S.MCUX_EXPERIMENTAL_MACRO (*C macro*), 542
drivers_RT_RT700_MIMXRT798S.MCUX_MASK_INVERT_8 (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.MCUX_MASK_INVERT_16 (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.MCUX_MASK_INVERT_32 (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.MCUX_RAMFUNC (*C macro*), 544
drivers_RT_RT700_MIMXRT798S.MCUX_REG_BIT_CLEAR8 (*C macro*), 540

drivers_RT_RT700_MIMXRT798S.MCUX_REG_BIT_CLEAR16 (*C macro*), 540
drivers_RT_RT700_MIMXRT798S.MCUX_REG_BIT_CLEAR32 (*C macro*), 540
drivers_RT_RT700_MIMXRT798S.MCUX_REG_BIT_GET8 (*C macro*), 540
drivers_RT_RT700_MIMXRT798S.MCUX_REG_BIT_GET16 (*C macro*), 540
drivers_RT_RT700_MIMXRT798S.MCUX_REG_BIT_GET32 (*C macro*), 540
drivers_RT_RT700_MIMXRT798S.MCUX_REG_BIT_SET8 (*C macro*), 540
drivers_RT_RT700_MIMXRT798S.MCUX_REG_BIT_SET16 (*C macro*), 540
drivers_RT_RT700_MIMXRT798S.MCUX_REG_BIT_SET32 (*C macro*), 540
drivers_RT_RT700_MIMXRT798S.MCUX_REG_MODIFY8 (*C macro*), 540
drivers_RT_RT700_MIMXRT798S.MCUX_REG_MODIFY16 (*C macro*), 540
drivers_RT_RT700_MIMXRT798S.MCUX_REG_MODIFY32 (*C macro*), 540
drivers_RT_RT700_MIMXRT798S.MCUX_REG_READ8 (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.MCUX_REG_READ16 (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.MCUX_REG_READ32 (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.MCUX_REG_WRITE8 (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.MCUX_REG_WRITE16 (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.MCUX_REG_WRITE32 (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.MCUX_SDK_PDM_EDMA_PDM_ENABLE_INTERNAL (*C macro*), 774
drivers_RT_RT700_MIMXRT798S.MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK (*C macro*), 866
drivers_RT_RT700_MIMXRT798S.MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG (*C macro*), 866
drivers_RT_RT700_MIMXRT798S.MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL (*C macro*), 877
drivers_RT_RT700_MIMXRT798S.MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL (*C macro*), 877
drivers_RT_RT700_MIMXRT798S.MD2BASECLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.MDNBASECLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.MEDIAMAINCLKDIV_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.MEDIAMAINCLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.MEDIAVDDNCLKDIV_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.MEDIAVDDNCLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.MICFIL0FCLKDIV_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.MICFIL0FCLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.MIN (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.MIPI_DSI_HOST_CLOCKS (*C macro*), 258
drivers_RT_RT700_MIMXRT798S.MIPI_DSI_RSTS (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.mmu_ahb_access_type_t (*C type*), 732
drivers_RT_RT700_MIMXRT798S.mmu_ahb_error_record_t (*C type*), 732
drivers_RT_RT700_MIMXRT798S.mmu_ahb_error_type_t (*C type*), 732
drivers_RT_RT700_MIMXRT798S.MMU_ChangePageOption (*C function*), 725
drivers_RT_RT700_MIMXRT798S.MMU_CheckMMUBusy (*C function*), 728
drivers_RT_RT700_MIMXRT798S.MMU_ClearStatusFlags (*C function*), 728
drivers_RT_RT700_MIMXRT798S.MMU_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.mmu_config_t (*C type*), 732
drivers_RT_RT700_MIMXRT798S.MMU_Deinit (*C function*), 722
drivers_RT_RT700_MIMXRT798S.MMU_DisableInterrupts (*C function*), 729
drivers_RT_RT700_MIMXRT798S.MMU_DoSoftwareReset (*C function*), 725
drivers_RT_RT700_MIMXRT798S.MMU_EnableInterrupts (*C function*), 728
drivers_RT_RT700_MIMXRT798S.MMU_EnableMemoryMap (*C function*), 724
drivers_RT_RT700_MIMXRT798S.MMU_GetAHBErrorRecord (*C function*), 728
drivers_RT_RT700_MIMXRT798S.MMU_GetDefaultConfig (*C function*), 722
drivers_RT_RT700_MIMXRT798S.MMU_GetIPSReadErrorAddress (*C function*), 728
drivers_RT_RT700_MIMXRT798S.MMU_GetPhyAddr (*C function*), 724
drivers_RT_RT700_MIMXRT798S.MMU_GetStatusFlags (*C function*), 727
drivers_RT_RT700_MIMXRT798S.MMU_Init (*C function*), 722
drivers_RT_RT700_MIMXRT798S.mmu_invalid_address_mode_t (*C type*), 732

drivers_RT_RT700_MIMXRT798S.MMU_LockLUT (*C function*), 727
drivers_RT_RT700_MIMXRT798S.mmu_lut_word_t (*C type*), 732
drivers_RT_RT700_MIMXRT798S.mmu_lut_write_operation_t (*C type*), 732
drivers_RT_RT700_MIMXRT798S.MMU_MoveArray (*C function*), 724
drivers_RT_RT700_MIMXRT798S.MMU_MoveRegion (*C function*), 723
drivers_RT_RT700_MIMXRT798S.MMU_NUM_OF_UNIT_SIZE (*C macro*), 733
drivers_RT_RT700_MIMXRT798S.MMU_PAGE_OPT (*C macro*), 733
drivers_RT_RT700_MIMXRT798S.MMU_PAGE_OPT_MASK (*C macro*), 733
drivers_RT_RT700_MIMXRT798S.MMU_PAGE_OPT_SHIFT (*C macro*), 733
drivers_RT_RT700_MIMXRT798S.mmu_page_option_t (*C type*), 732
drivers_RT_RT700_MIMXRT798S.MMU_ReadLUT (*C function*), 726
drivers_RT_RT700_MIMXRT798S.MMU_RemapRegion (*C function*), 723
drivers_RT_RT700_MIMXRT798S.MMU_SetupArrayMap (*C function*), 723
drivers_RT_RT700_MIMXRT798S.MMU_SetupRegionMap (*C function*), 723
drivers_RT_RT700_MIMXRT798S.MMU_SetupSimpleMap (*C function*), 722
drivers_RT_RT700_MIMXRT798S.MMU_TearDownArrayMap (*C function*), 724
drivers_RT_RT700_MIMXRT798S.MMU_TearDownRegionMap (*C function*), 723
drivers_RT_RT700_MIMXRT798S.MMU_TearDownSimpleMap (*C function*), 723
drivers_RT_RT700_MIMXRT798S.MMU_TriggerLUTVerificationBlocking (*C function*), 727
drivers_RT_RT700_MIMXRT798S.mmu_unit_size_t (*C type*), 732
drivers_RT_RT700_MIMXRT798S.MMU_UnlockLUT (*C function*), 727
drivers_RT_RT700_MIMXRT798S.MMU_UpdateLUTBlocking (*C function*), 725
drivers_RT_RT700_MIMXRT798S.MMU_UpdateUnitSize (*C function*), 725
drivers_RT_RT700_MIMXRT798S.MMU_UpdateValidAddressRange (*C function*), 725
drivers_RT_RT700_MIMXRT798S.mode (*C var*), 842
drivers_RT_RT700_MIMXRT798S.modefunc (*C var*), 516
drivers_RT_RT700_MIMXRT798S.monitorEnable (*C var*), 267
drivers_RT_RT700_MIMXRT798S.mrt_chnl_t (*C type*), 738
drivers_RT_RT700_MIMXRT798S.MRT_ClearStatusFlags (*C function*), 736
drivers_RT_RT700_MIMXRT798S.MRT_CLOCKS (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.mrt_config_t (*C type*), 738
drivers_RT_RT700_MIMXRT798S.MRT_Deinit (*C function*), 734
drivers_RT_RT700_MIMXRT798S.MRT_DisableInterrupts (*C function*), 735
drivers_RT_RT700_MIMXRT798S.MRT_EnableInterrupts (*C function*), 735
drivers_RT_RT700_MIMXRT798S.MRT_GetCurrentTimerCount (*C function*), 736
drivers_RT_RT700_MIMXRT798S.MRT_GetDefaultConfig (*C function*), 734
drivers_RT_RT700_MIMXRT798S.MRT_GetEnabledInterrupts (*C function*), 735
drivers_RT_RT700_MIMXRT798S.MRT_GetIdleChannel (*C function*), 737
drivers_RT_RT700_MIMXRT798S.MRT_GetStatusFlags (*C function*), 735
drivers_RT_RT700_MIMXRT798S.MRT_Init (*C function*), 734
drivers_RT_RT700_MIMXRT798S.mrt_interrupt_enable_t (*C type*), 738
drivers_RT_RT700_MIMXRT798S.MRT_ReleaseChannel (*C function*), 737
drivers_RT_RT700_MIMXRT798S.MRT_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.MRT_SetupChannelMode (*C function*), 735
drivers_RT_RT700_MIMXRT798S.MRT_StartTimer (*C function*), 736
drivers_RT_RT700_MIMXRT798S.mrt_status_flags_t (*C type*), 738
drivers_RT_RT700_MIMXRT798S.MRT_StopTimer (*C function*), 737
drivers_RT_RT700_MIMXRT798S.mrt_timer_mode_t (*C type*), 738
drivers_RT_RT700_MIMXRT798S.MRT_UpdateTimerPeriod (*C function*), 736
drivers_RT_RT700_MIMXRT798S.MSDK_HAS_DWT_CYCCNT (*C macro*), 557
drivers_RT_RT700_MIMXRT798S.MSDK_INVALID_IRQ_HANDLER (*C macro*), 557
drivers_RT_RT700_MIMXRT798S.MSDK_REG_NONSECURE_ADDR (*C macro*), 556, 557
drivers_RT_RT700_MIMXRT798S.MSDK_REG_SECURE_ADDR (*C macro*), 556, 557
drivers_RT_RT700_MIMXRT798S.MSEC_TO_COUNT (*C macro*), 540, 541
drivers_RT_RT700_MIMXRT798S.MU1_BUSY_POLL_COUNT (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_BootOtherCore (*C function*), 746
drivers_RT_RT700_MIMXRT798S.MU_ClearGeneralPurposeStatusFlags (*C function*), 745
drivers_RT_RT700_MIMXRT798S.MU_ClearStatusFlags (*C function*), 743

drivers_RT_RT700_MIMXRT798S.MU_CORE_FLAG (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_CORE_INTR (*C macro*), 751
drivers_RT_RT700_MIMXRT798S.MU_Deinit (*C function*), 739
drivers_RT_RT700_MIMXRT798S.MU_DisableGeneralPurposeInterrupts (*C function*), 744
drivers_RT_RT700_MIMXRT798S.MU_DisableInterrupts (*C function*), 743
drivers_RT_RT700_MIMXRT798S.MU_EnableGeneralPurposeInterrupts (*C function*), 744
drivers_RT_RT700_MIMXRT798S.MU_EnableInterrupts (*C function*), 743
drivers_RT_RT700_MIMXRT798S.mu_general_purpose_interrupt_t (*C type*), 751
drivers_RT_RT700_MIMXRT798S.MU_GET_CORE_FLAG (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_GET_CORE_INTR (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_GET_GI_FLAG (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_GET_GI_INTR (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_GET_RX_FLAG (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_GET_RX_INTR (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_GET_STAT_FLAG (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_GET_TX_FLAG (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_GET_TX_INTR (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_GetFlags (*C function*), 742
drivers_RT_RT700_MIMXRT798S.MU_GetGeneralPurposeStatusFlags (*C function*), 745
drivers_RT_RT700_MIMXRT798S.MU_GetInterruptsPending (*C function*), 742
drivers_RT_RT700_MIMXRT798S.MU_GetRxStatusFlags (*C function*), 745
drivers_RT_RT700_MIMXRT798S.MU_GetStatusFlags (*C function*), 742
drivers_RT_RT700_MIMXRT798S.MU_GI_FLAG (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_GI_INTR (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_HardwareResetOtherCore (*C function*), 747
drivers_RT_RT700_MIMXRT798S.MU_HoldOtherCoreReset (*C function*), 746
drivers_RT_RT700_MIMXRT798S.MU_Init (*C function*), 739
drivers_RT_RT700_MIMXRT798S.MU_MISC_INTR (*C macro*), 751
drivers_RT_RT700_MIMXRT798S.mu_msg_reg_index_t (*C type*), 751
drivers_RT_RT700_MIMXRT798S.MU_ReceiveMsg (*C function*), 741
drivers_RT_RT700_MIMXRT798S.MU_ReceiveMsgNonBlocking (*C function*), 740
drivers_RT_RT700_MIMXRT798S.MU_ReceiveMsgTimeout (*C function*), 740
drivers_RT_RT700_MIMXRT798S.MU_ResetBothSides (*C function*), 747
drivers_RT_RT700_MIMXRT798S.MU_RX_FLAG (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_RX_INTR (*C macro*), 751
drivers_RT_RT700_MIMXRT798S.MU_SendMsg (*C function*), 739
drivers_RT_RT700_MIMXRT798S.MU_SendMsgNonBlocking (*C function*), 739
drivers_RT_RT700_MIMXRT798S.MU_SetFlags (*C function*), 741
drivers_RT_RT700_MIMXRT798S.MU_SetFlagsNonBlocking (*C function*), 741
drivers_RT_RT700_MIMXRT798S.MU_STAT_FLAG (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_TriggerGeneralPurposeInterrupts (*C function*), 746
drivers_RT_RT700_MIMXRT798S.MU_TriggerInterrupts (*C function*), 744
drivers_RT_RT700_MIMXRT798S.MU_TX_FLAG (*C macro*), 752
drivers_RT_RT700_MIMXRT798S.MU_TX_INTR (*C macro*), 751
drivers_RT_RT700_MIMXRT798S.nbytes (*C var*), 1051
drivers_RT_RT700_MIMXRT798S.nextDesctAddr (*C var*), 537
drivers_RT_RT700_MIMXRT798S.nsize (*C var*), 1050
drivers_RT_RT700_MIMXRT798S.numerator (*C var*), 266, 267
drivers_RT_RT700_MIMXRT798S.NVIC_ClearPendingIRQ (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.NVIC_DisableIRQ (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.NVIC_EnableIRQ (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.NVIC_GetActive (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.NVIC_GetEnableIRQ (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.NVIC_GetPendingIRQ (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.NVIC_GetPriority (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.NVIC_GetPriorityGrouping (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.NVIC_SetPendingIRQ (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.NVIC_SetPriority (*C macro*), 556

drivers_RT_RT700_MIMXRT798S.NVIC_SetPriorityGrouping (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.option (*C var*), 842
drivers_RT_RT700_MIMXRT798S.osc32k_cap_trim_t (*C type*), 246
drivers_RT_RT700_MIMXRT798S.OSEVENTFCLKDIV_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.OSEVENTTFCLKSEL_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.ostimer_callback_t (*C type*), 758
drivers_RT_RT700_MIMXRT798S.OSTIMER_ClearStatusFlags (*C function*), 754
drivers_RT_RT700_MIMXRT798S.OSTIMER_CLOCKS (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.OSTIMER_DecimalToGray (*C function*), 754
drivers_RT_RT700_MIMXRT798S.OSTIMER_Deinit (*C function*), 753
drivers_RT_RT700_MIMXRT798S.OSTIMER_DisableMatchInterrupt (*C function*), 756
drivers_RT_RT700_MIMXRT798S.OSTIMER_DriverIRQHandler (*C function*), 758
drivers_RT_RT700_MIMXRT798S.OSTIMER_EnableMatchInterrupt (*C function*), 756
drivers_RT_RT700_MIMXRT798S.OSTIMER_GetCaptureRawValue (*C function*), 757
drivers_RT_RT700_MIMXRT798S.OSTIMER_GetCaptureValue (*C function*), 757
drivers_RT_RT700_MIMXRT798S.OSTIMER_GetCurrentTimerRawValue (*C function*), 757
drivers_RT_RT700_MIMXRT798S.OSTIMER_GetCurrentTimerValue (*C function*), 757
drivers_RT_RT700_MIMXRT798S.OSTIMER_GetMatchRegister (*C function*), 756
drivers_RT_RT700_MIMXRT798S.OSTIMER_GetMatchValue (*C function*), 756
drivers_RT_RT700_MIMXRT798S.OSTIMER_GetStatusFlags (*C function*), 754
drivers_RT_RT700_MIMXRT798S.OSTIMER_GrayToDecimal (*C function*), 753
drivers_RT_RT700_MIMXRT798S.OSTIMER_HandleIRQ (*C function*), 757
drivers_RT_RT700_MIMXRT798S.OSTIMER_Init (*C function*), 753
drivers_RT_RT700_MIMXRT798S.OSTIMER_SetMatchRawValue (*C function*), 754
drivers_RT_RT700_MIMXRT798S.OSTIMER_SetMatchRegister (*C function*), 756
drivers_RT_RT700_MIMXRT798S.OSTIMER_SetMatchValue (*C function*), 754
drivers_RT_RT700_MIMXRT798S.OSTIMER_SetMatchValueSafe (*C function*), 755
drivers_RT_RT700_MIMXRT798S.OUT_ACTIONS_MASK (*C macro*), 529
drivers_RT_RT700_MIMXRT798S.outBufAddr0 (*C var*), 536
drivers_RT_RT700_MIMXRT798S.outBufAddr1 (*C var*), 536
drivers_RT_RT700_MIMXRT798S.outBufPitch (*C var*), 536
drivers_RT_RT700_MIMXRT798S.pd_bit_t (*C type*), 806
drivers_RT_RT700_MIMXRT798S.pd_bits (*C enum*), 791
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_CPU0_CODE_CACHE (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_CPU0_SYSTEM_CACHE (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_DMA0_1_PKC_ETF (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_DMA2_3 (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_DSP_DCACHE (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_DSP_DTCM (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_DSP_ICACHE (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_DSP_ITCM (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_EZHV_TCM (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_GPU (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_JPEGDEC (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_LCDIF (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_MIPIDSI (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_NPU (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_OCOTP (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_PNGDEC (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SDHC0_SRAM (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SDHC1_SRAM (*C enumerator*), 797

drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM1 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM2 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM3 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM4 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM5 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM6 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM7 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM8 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM9 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM10 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM11 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM12 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM13 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM14 (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM15 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM16 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM17 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM18 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM19 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM20 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM21 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM22 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM23 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM24 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM25 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM26 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM27 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM28 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_SRAM29 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_USB0_SRAM (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_USB1_SRAM (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_XSPI0 (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_XSPI1 (*C enumerator*), 798
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_APD_XSPI2 (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_DSR_VDD2N_MEDIA (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_DSR_VDDN_COM (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_ForceUnsigned (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_GATE_FRO2 (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_LP_LVD_VDD1 (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_LP_LVD_VDD2 (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_LP_LVD_VDDN (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_LP_PMCREF (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_LP_POR_VDD1 (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_LP_POR_VDD2 (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_LP_POR_VDDN (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_ADC0 (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_AFBB_VDD1 (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_AFBB_VDD2 (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_AFBB_VDDN (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_AGDET_VDD1 (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_AGDET_VDD2 (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_AUDPLLANA (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_AUDPLLLDO (*C enumerator*), 792

drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_FRO2 (*C enumerator*), 791
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_HVD1V8 (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_HVD_VDD1 (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_HVD_VDD2 (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_HVD_VDDN (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_LPOSC (*C enumerator*), 791
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_OTP (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_PLLANA (*C enumerator*), 791
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_PLLLDO (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_PMC_TEMPSNS (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_RBB_VDD1 (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_RBB_VDD2 (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_RBB_VDDN (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_ROM (*C enumerator*), 794
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_SRAM_AFBB_VDD2 (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_SRAM_RBB_VDD1 (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_SRAM_RBB_VDD2 (*C enumerator*), 793
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_SYSXTAL (*C enumerator*), 791
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_VDD2_DSP (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PD_VDD2_MIPI (*C enumerator*), 792
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_CPU0_CODE_CACHE (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_CPU0_SYSTEM_CACHE (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_DMA0_1_PKC_ETF (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_DMA2_3 (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_DSP_DCACHE (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_DSP_DTCM (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_DSP_ICACHE (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_DSP_ITCM (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_EZHV_TCM (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_GPU (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_JPEGDEC (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_LCDIF (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_MIPIDSI (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_NPU (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_OCOTP (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_PNGDEC (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SDHC0_SRAM (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SDHC1_SRAM (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM1 (*C enumerator*), 795
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM2 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM3 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM4 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM5 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM6 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM7 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM8 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM9 (*C enumerator*), 796

drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM10 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM11 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM12 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM13 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM14 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM15 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM16 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM17 (*C enumerator*), 796
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM18 (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM19 (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM20 (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM21 (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM22 (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM23 (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM24 (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM25 (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM26 (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM27 (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM28 (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_SRAM29 (*C enumerator*), 797
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_USB0_SRAM (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_USB1_SRAM (*C enumerator*), 799
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_XSPI0 (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_XSPI1 (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_PPD_XSPI2 (*C enumerator*), 800
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_SHUT_COMNN_MAINCLK (*C enumerator*), 791
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_SHUT_MEDIA_MAINCLK (*C enumerator*), 791
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_SHUT_RAM0_CLK (*C enumerator*), 791
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_SHUT_RAM1_CLK (*C enumerator*), 791
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_SHUT_SENSEP_MAINCLK (*C enumerator*), 791
drivers_RT_RT700_MIMXRT798S.pd_bits.kPDRUNCFG_SHUT_SENSES_MAINCLK (*C enumerator*), 791
drivers_RT_RT700_MIMXRT798S.pdm_channel_config_t (*C type*), 769
drivers_RT_RT700_MIMXRT798S.PDM_ClearFIFOStatus (*C function*), 761
drivers_RT_RT700_MIMXRT798S.PDM_ClearRangeStatus (*C function*), 761
drivers_RT_RT700_MIMXRT798S.PDM_ClearStatus (*C function*), 761
drivers_RT_RT700_MIMXRT798S.PDM_CLOCKS (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.pdm_config_t (*C type*), 769
drivers_RT_RT700_MIMXRT798S.pdm_dc_remover_t (*C type*), 769
drivers_RT_RT700_MIMXRT798S.PDM_Deinit (*C function*), 758
drivers_RT_RT700_MIMXRT798S.pdm_df_output_gain_t (*C type*), 769
drivers_RT_RT700_MIMXRT798S.pdm_df_quality_mode_t (*C type*), 769
drivers_RT_RT700_MIMXRT798S.PDM_DisableInterrupts (*C function*), 761
drivers_RT_RT700_MIMXRT798S.pdm_edma_callback_t (*C type*), 774
drivers_RT_RT700_MIMXRT798S.pdm_edma_handle_t (*C type*), 774
drivers_RT_RT700_MIMXRT798S.pdm_edma_multi_channel_interleave_t (*C type*), 774
drivers_RT_RT700_MIMXRT798S.pdm_edma_transfer_t (*C type*), 774
drivers_RT_RT700_MIMXRT798S.PDM_Enable (*C function*), 759
drivers_RT_RT700_MIMXRT798S.PDM_EnableChannel (*C function*), 759
drivers_RT_RT700_MIMXRT798S.PDM_EnableDebugMode (*C function*), 759
drivers_RT_RT700_MIMXRT798S.PDM_EnableDMA (*C function*), 762

drivers_RT_RT700_MIMXRT798S.PDM_EnableInDebugMode (*C function*), 759
drivers_RT_RT700_MIMXRT798S.PDM_EnableInterrupts (*C function*), 761
drivers_RT_RT700_MIMXRT798S.PDM_EnterLowLeakageMode (*C function*), 759
drivers_RT_RT700_MIMXRT798S.PDM_GetDataRegisterAddress (*C function*), 762
drivers_RT_RT700_MIMXRT798S.PDM_GetFifoStatus (*C function*), 760
drivers_RT_RT700_MIMXRT798S.PDM_GetInstance (*C function*), 760
drivers_RT_RT700_MIMXRT798S.PDM_GetRangeStatus (*C function*), 761
drivers_RT_RT700_MIMXRT798S.PDM_GetStatus (*C function*), 760
drivers_RT_RT700_MIMXRT798S.pdm_handle_t (*C type*), 769
drivers_RT_RT700_MIMXRT798S.PDM_Init (*C function*), 758
drivers_RT_RT700_MIMXRT798S.PDM_ReadFifo (*C function*), 762
drivers_RT_RT700_MIMXRT798S.PDM_Reset (*C function*), 759
drivers_RT_RT700_MIMXRT798S.PDM_SetChannelConfig (*C function*), 759
drivers_RT_RT700_MIMXRT798S.PDM_SetChannelGain (*C function*), 762
drivers_RT_RT700_MIMXRT798S.PDM_SetSampleRate (*C function*), 760
drivers_RT_RT700_MIMXRT798S.PDM_SetSampleRateConfig (*C function*), 760
drivers_RT_RT700_MIMXRT798S.pdm_transfer_callback_t (*C type*), 769
drivers_RT_RT700_MIMXRT798S.pdm_transfer_t (*C type*), 769
drivers_RT_RT700_MIMXRT798S.PDM_TransferAbortReceive (*C function*), 763
drivers_RT_RT700_MIMXRT798S.PDM_TransferAbortReceiveEDMA (*C function*), 773
drivers_RT_RT700_MIMXRT798S.PDM_TransferCreateHandle (*C function*), 762
drivers_RT_RT700_MIMXRT798S.PDM_TransferCreateHandleEDMA (*C function*), 771
drivers_RT_RT700_MIMXRT798S.PDM_TransferGetReceiveCountEDMA (*C function*), 773
drivers_RT_RT700_MIMXRT798S.PDM_TransferHandleIRQ (*C function*), 764
drivers_RT_RT700_MIMXRT798S.PDM_TransferInstallEDMATCDMemory (*C function*), 771
drivers_RT_RT700_MIMXRT798S.PDM_TransferReceiveEDMA (*C function*), 772
drivers_RT_RT700_MIMXRT798S.PDM_TransferReceiveNonBlocking (*C function*), 763
drivers_RT_RT700_MIMXRT798S.PDM_TransferSetChannelConfig (*C function*), 763
drivers_RT_RT700_MIMXRT798S.PDM_TransferSetChannelConfigEDMA (*C function*), 772
drivers_RT_RT700_MIMXRT798S.PDM_TransferSetMultiChannelInterleaveType (*C function*), 772
drivers_RT_RT700_MIMXRT798S.PDM_TransferTerminateReceiveEDMA (*C function*), 773
drivers_RT_RT700_MIMXRT798S.PDM_XFER_QUEUE_SIZE (*C macro*), 769
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_ApbNotIdle (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_ApbRxControl (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_ApbRxHeaderReceived (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_ApbRxOverflow (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_ApbRxPacketReceived (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_ApbRxUnderflow (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_ApbTxDone (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_ApbTxOverflow (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_ApbTxUnderflow (*C enumerator*), 709
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_ClockLaneUlpsEnable (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_DataLane0UlpsEnable (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_DataLane1UlpsEnable (*C enumerator*), 711
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_DataLane2UlpsEnable (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_DataLane3UlpsEnable (*C enumerator*), 712

drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_DpiHsyncActiveHigh (*C enumerator*), 708
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_DpiHsyncActiveLow (*C enumerator*), 708
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_DpiVsyncActiveHigh (*C enumerator*), 708
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_DpiVsyncActiveLow (*C enumerator*), 708
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1AckTriggerReceived (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ApbNotIdle (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ApbRxControl (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ApbRxHeaderReceived (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ApbRxOverflow (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ApbRxPacketReceived (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ApbRxUnderflow (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ApbTxDone (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ApbTxOverflow (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ApbTxUnderflow (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1BtaTo (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ChecksumError (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ContentionDetected (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1EccErrorMultiBit (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1EccErrorOneBit (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1EoTSyncError (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1EscEntryCmdError (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1FalseControlError (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1HtxTo (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1InvalidDataType (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1InvalidTxLength (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1InvalidVcId (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1LpTxSyncError (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1LrxTo (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1PeriphToError (*C enumerator*), 713

drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ProtocalViolation (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1ResetTriggerReceived (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1SoTError (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1SoTSyncError (*C enumerator*), 712
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup1TearTriggerReceived (*C enumerator*), 713
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup2CrcError (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup2EccMultiBit (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_InterruptGroup2EccOneBit (*C enumerator*), 714
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_RxErrorBtaTo (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_RxErrorCrc (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_RxErrorEccMultiBit (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_RxErrorEccOneBit (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_RxErrorHtxTo (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_RxErrorLrxTo (*C enumerator*), 710
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_TransferPerformBTA (*C enumerator*), 717
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kDSI_TransferUseHighSpeed (*C enumerator*), 717
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_DestinationAddressErrorFlag (*C enumerator*), 306
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_DestinationBusErrorFlag (*C enumerator*), 305
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_DestinationOffsetErrorFlag (*C enumerator*), 306
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_DoneFlag (*C enumerator*), 305
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_ErrorChannelFlag (*C enumerator*), 306
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_ErrorFlag (*C enumerator*), 305
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_ErrorInterruptEnable (*C enumerator*), 306
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_HalfInterruptEnable (*C enumerator*), 306
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_InterruptFlag (*C enumerator*), 305
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_MajorInterruptEnable (*C enumerator*), 306
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_MasterId (*C enumerator*), 308
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_NbytesErrorFlag (*C enumerator*), 305
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_PrivilegedAccessLevel (*C enumerator*), 308
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_ScatterGatherErrorFlag (*C enumerator*), 305
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_SourceAddressErrorFlag (*C enumerator*), 306

drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_SourceBusErrorFlag (*C enumerator*), 305
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_SourceOffsetErrorFlag (*C enumerator*), 306
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_TransferCanceledFlag (*C enumerator*), 306
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kEDMA_ValidFlag (*C enumerator*), 306
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFlashInstMode_0_4_4_DDR (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFlashInstMode_0_4_4_SDR (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFlashInstMode_DPI_DDR (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFlashInstMode_DPI_SDR (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFlashInstMode_ExtendedSpi (*C enumerator*), 836
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFlashInstMode_OPI_DDR (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFlashInstMode_OPI_SDR (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFlashInstMode_QPI_DDR (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFlashInstMode_QPI_SDR (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFLEXIO_I2S_RxDataRegFullFlag (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFLEXIO_I2S_RxDataRegFullInterruptEnable (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFLEXIO_I2S_TxDataRegEmptyFlag (*C enumerator*), 384
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFLEXIO_I2S_TxDataRegEmptyInterruptEnable (*C enumerator*), 383
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFwVerChk_Id_none (*C enumerator*), 879
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFwVerChk_Id_nonsecure (*C enumerator*), 879
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kFwVerChk_Id_secure (*C enumerator*), 879
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kLCDIF_AlphaFactorInversed (*C enumerator*), 570
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kLCDIF_AlphaFactorOne (*C enumerator*), 570
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kLCDIF_AlphaFactorStraight (*C enumerator*), 570
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kLCDIF_AlphaFactorZero (*C enumerator*), 570
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kLCDIF_AlphaGlobal (*C enumerator*), 570
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kLCDIF_AlphaInversed (*C enumerator*), 569
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kLCDIF_AlphaLocal (*C enumerator*), 569
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kLCDIF_AlphaScaled (*C enumerator*), 570
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kLCDIF_AlphaStraight (*C enumerator*), 569
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kRestoreSeqence_None (*C enumerator*),

837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kRestoreSeqence_QPI_4_0xFFs (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kRestoreSeqence_QPI_5_0xFFs (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kRestoreSeqence_QPI_Mode_0x00 (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kRestoreSeqence_Send_06_FF (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kRestoreSeqence_Send_66_99 (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kRestoreSeqence_Send_6699_9966 (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kRestoreSeqence_Send_F0 (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kRestoreSeqence_Send_QPI_8_0xFFs (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kRestoreSeqence_Wakep_0xAB (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kRestoreSeqence_Wakep_0xAB_54 (*C enumerator*), 837
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_Channel0Mask (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_Channel1Mask (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_Channel2Mask (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_Channel3Mask (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_Channel4Mask (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_Channel5Mask (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_Channel6Mask (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_Channel7Mask (*C enumerator*), 858
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_FIFOErrorFlag (*C enumerator*), 861
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_FIFOErrorInterruptEnable (*C enumerator*), 861
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_FIFORequestDMAEnable (*C enumerator*), 861
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_FIFORequestFlag (*C enumerator*), 861
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_FIFORequestInterruptEnable (*C enumerator*), 861
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_FIFOWarningDMAEnable (*C enumerator*), 861
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_FIFOWarningFlag (*C enumerator*), 861
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_FIFOWarningInterruptEnable (*C enumerator*), 861
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_SyncErrorFlag (*C enumerator*), 861
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_SyncErrorInterruptEnable (*C enumerator*), 861
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_WordStartFlag (*C enumerator*),

861
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kSAI_WordStartInterruptEnable (*C enumerator*), 861
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_Busy (*C enumerator*), 553
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_DSI_Busy (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_DSI_ErrorReportReceived (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_DSI_NotSupported (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_DSI_RxDataError (*C enumerator*), 707
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_EDMA_Busy (*C enumerator*), 302
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_EDMA_QueueFull (*C enumerator*), 302
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_Fail (*C enumerator*), 553
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_CAMERA_RxBusy (*C enumerator*), 332
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_CAMERA_RxIdle (*C enumerator*), 332
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_I2C_Busy (*C enumerator*), 373
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_I2C_Idle (*C enumerator*), 373
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_I2C_Nak (*C enumerator*), 373
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_I2C_Timeout (*C enumerator*), 373
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_I2S_Error (*C enumerator*), 383
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_I2S_Idle (*C enumerator*), 383
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_I2S_QueueFull (*C enumerator*), 383
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_I2S_RxBusy (*C enumerator*), 383
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_I2S_Timeout (*C enumerator*), 383
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_I2S_TxBusy (*C enumerator*), 383
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_MCULCD_Busy (*C enumerator*), 396
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_MCULCD_Error (*C enumerator*), 396
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_MCULCD_Idle (*C enumerator*), 396
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_SPI_Busy (*C enumerator*), 409
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_SPI_Error (*C enumerator*), 409
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_SPI_Idle (*C enumerator*), 409
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_SPI_Timeout (*C enumerator*), 409
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_UART_BaudrateNotSupport (*C enumerator*), 423
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_UART_ERROR (*C enumerator*), 423

drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_UART_RxBusy (*C enumerator*), 423
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_UART_RxHardwareOverrun (*C enumerator*), 423
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_UART_RxIdle (*C enumerator*), 423
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_UART_RxRingBufferOverrun (*C enumerator*), 423
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_UART_Timeout (*C enumerator*), 423
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_UART_TxBusy (*C enumerator*), 422
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_FLEXIO_UART_TxIdle (*C enumerator*), 423
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_GLIKEY_DisabledError (*C enumerator*), 429
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_GLIKEY_Locked (*C enumerator*), 429
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_GLIKEY_LockedError (*C enumerator*), 429
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_GLIKEY_NotLocked (*C enumerator*), 429
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_Busy (*C enumerator*), 443
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_CrcError (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_HdrParityError (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_IBIWon (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_Idle (*C enumerator*), 443
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_InvalidReq (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_InvalidStart (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_MsgError (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_Nak (*C enumerator*), 443
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_OverrunError (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_ReadFifoError (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_S0S1Error (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_SdrParityError (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_SlaveCountExceed (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_Term (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_Timeout (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_UnderrunError (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_UnderrunNak (*C enumerator*), 444
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_WriteAbort (*C enumerator*), 443
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_I3C_WriteFifoError (*C enumerator*), 444

drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_InvalidArgument (*C enumerator*), 553
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_JPEGDEC_NotSupported (*C enumerator*), 530
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPI2C_ArbitrationLost (*C enumerator*), 612
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPI2C_BitError (*C enumerator*), 612
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPI2C_Busy (*C enumerator*), 612
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPI2C_DmaRequestFail (*C enumerator*), 612
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPI2C_FifoError (*C enumerator*), 612
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPI2C_Idle (*C enumerator*), 612
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPI2C_Nak (*C enumerator*), 612
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPI2C_NoTransferInProgress (*C enumerator*), 612
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPI2C_PinLowTimeout (*C enumerator*), 612
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPI2C_Timeout (*C enumerator*), 612
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPSPI_Busy (*C enumerator*), 652
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPSPI_Error (*C enumerator*), 652
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPSPI_Idle (*C enumerator*), 652
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPSPI_OutOfRange (*C enumerator*), 652
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPSPI_Timeout (*C enumerator*), 652
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_BaudrateNotSupport (*C enumerator*), 684
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_Error (*C enumerator*), 684
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_FlagCannotClearManually (*C enumerator*), 684
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_FramingError (*C enumerator*), 684
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_IdleLineDetected (*C enumerator*), 684
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_NoiseError (*C enumerator*), 684
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_ParityError (*C enumerator*), 684
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_RxBusy (*C enumerator*), 683
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_RxHardwareOverrun (*C enumerator*), 684
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_RxIdle (*C enumerator*), 683
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_RxRingBufferOverrun (*C enumerator*), 684
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_RxWatermarkTooLarge (*C enumerator*), 684

drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_Timeout (*C enumerator*), 684
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_TxBusy (*C enumerator*), 683
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_TxIdle (*C enumerator*), 683
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_LPUART_TxWatermarkTooLarge (*C enumerator*), 684
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_NoData (*C enumerator*), 553
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_NoTransferInProgress (*C enumerator*), 553
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_OutOfRange (*C enumerator*), 553
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PDM_Busy (*C enumerator*), 764
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PDM_ChannelConfig_Failed (*C enumerator*), 764
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PDM_CLK_LOW (*C enumerator*), 764
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PDM_FIFO_ERROR (*C enumerator*), 764
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PDM_Idle (*C enumerator*), 764
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PDM_Output_ERROR (*C enumerator*), 764
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PDM_QueueFull (*C enumerator*), 764
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PNGDEC_NotSupported (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PNGDEC_WidthTooLarge (*C enumerator*), 786
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PUF_AcAuthFailedPhase1 (*C enumerator*), 820
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PUF_AcCorruptedPhase1 (*C enumerator*), 820
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PUF_AcCorruptedPhase2 (*C enumerator*), 820
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PUF_AcNotForThisProductPhase1 (*C enumerator*), 820
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PUF_AcNotForThisProductPhase2 (*C enumerator*), 820
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PUF_ContextIncorrect (*C enumerator*), 820
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PUF_DestinationNotAllowed (*C enumerator*), 821
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PUF_Failure (*C enumerator*), 821
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PUF_NBOOT_AcAuthFailedPhase2 (*C enumerator*), 820
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PUF_OperationNotAllowed (*C enumerator*), 820
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_PUF_QualityVerificationFail (*C enumerator*), 820
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_ReadOnly (*C enumerator*), 553
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_SAI_QueueFull (*C enumerator*), 857
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_SAI_RxBusy (*C enumerator*), 857
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_SAI_RxError (*C enumerator*),

857
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_SAI_RxIdle (C enumerator), 857
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_SAI_TxBusy (C enumerator), 857
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_SAI_TxError (C enumerator), 857
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_SAI_TxIdle (C enumerator), 857
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_SEMA42_Busy (C enumerator), 924
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_SEMA42_Reseting (C enumerator), 924
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_Success (C enumerator), 553
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_Timeout (C enumerator), 553
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_USDHC_BusyTransferring (C enumerator), 948
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_USDHC_DMADataAddrNotAlign (C enumerator), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_USDHC_NotSupport (C enumerator), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_USDHC_PrepareAdmaDescriptorFailed (C enumerator), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_USDHC_ReTuningRequest (C enumerator), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_USDHC_SendCommandFailed (C enumerator), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_USDHC_SendCommandSuccess (C enumerator), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_USDHC_TransferDataComplete (C enumerator), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_USDHC_TransferDataFailed (C enumerator), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_USDHC_TransferDMAComplete (C enumerator), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_USDHC_TuningError (C enumerator), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_AhbReadAccessAsserted (C enumerator), 980
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_AhbSubBufferFactorError (C enumerator), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_AhbWriteAccessAsserted (C enumerator), 980
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_ArbiterGranted (C enumerator), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_Busy (C enumerator), 980
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_FradCheckError (C enumerator), 980
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_IllegalInstructionError (C enumerator), 980
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_IpAccessAddrSettingInvalid (C enumerator), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_IPAccessAsserted (C enumerator), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_IpAccessIPCRInvalid (C enumerator), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_IpAccessNotGranted (C

enumerator), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_IpCommandtriggerError (*C enumerator*), 980
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_IpCommandUsageError (*C enumerator*), 980
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_IpReadFinished (*C enumerator*), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_IpsBusTransError (*C enumerator*), 980
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_IpWriteFinished (*C enumerator*), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_PageProgramWaitFlagAsserted (*C enumerator*), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_RegWriteLocked (*C enumerator*), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_RxBufferEntriesCountError (*C enumerator*), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_SequenceExecutionTimeout (*C enumerator*), 980
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_TgQueueWritingError (*C enumerator*), 980
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_TxBufferUnderrun (*C enumerator*), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kStatus_XSPI_WaterMarkIllegal (*C enumerator*), 981
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma1DescriptorActivity1Flag (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma1DescriptorActivity2Flag (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma1DescriptorEndFlag (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma1DescriptorInterrupFlag (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma1DescriptorTypeLink (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma1DescriptorTypeNop (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma1DescriptorTypeSetLength (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma1DescriptorTypeTransfer (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma1DescriptorValidFlag (*C enumerator*), 960
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma2DescriptorActivity1Flag (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma2DescriptorActivity2Flag (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma2DescriptorEndFlag (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma2DescriptorInterruptFlag (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma2DescriptorTypeLink (*C enumerator*), 962
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma2DescriptorTypeNop (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma2DescriptorTypeReserved (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma2DescriptorTypeTransfer

(*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Adma2DescriptorValidFlag (*C enumerator*), 961
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AdmaDescriptorErrorFlag (*C enumerator*), 956
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AdmaDescriptorMultipleFlag (*C enumerator*), 962
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AdmaDescriptorSingleFlag (*C enumerator*), 962
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AdmaErrorState (*C enumerator*), 956
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AdmaErrorStateChangeAddress (*C enumerator*), 956
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AdmaErrorStateFetchDescriptor (*C enumerator*), 956
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AdmaErrorStateInvalidDescriptor (*C enumerator*), 956
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AdmaErrorStateInvalidLength (*C enumerator*), 956
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AdmaErrorStateStopDma (*C enumerator*), 956
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AdmaErrorStateTransferData (*C enumerator*), 956
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AdmaLenghMismatchFlag (*C enumerator*), 956
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AllInterruptFlags (*C enumerator*), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AutoCommand12CrcErrorFlag (*C enumerator*), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AutoCommand12EndBitErrorFlag (*C enumerator*), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AutoCommand12ErrorFlag (*C enumerator*), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AutoCommand12IndexErrorFlag (*C enumerator*), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AutoCommand12NotExecutedFlag (*C enumerator*), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AutoCommand12NotIssuedFlag (*C enumerator*), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_AutoCommand12TimeoutFlag (*C enumerator*), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_BlockGapEventFlag (*C enumerator*), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_BufferReadEnableFlag (*C enumerator*), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_BufferReadReadyFlag (*C enumerator*), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_BufferWriteEnableFlag (*C enumerator*), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_BufferWriteReadyFlag (*C enumerator*), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CardDetectFlag (*C enumerator*), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CardInsertedFlag (*C enumerator*), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CardInsertionFlag (*C enumerator*), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CardInterruptFlag (*C enumer-*

ator), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CardRemovalFlag (C enumerator), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CommandCompleteFlag (C enumerator), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CommandCrcErrorFlag (C enumerator), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CommandEndBitErrorFlag (C enumerator), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CommandErrorFlag (C enumerator), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CommandFlag (C enumerator), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CommandIndexErrorFlag (C enumerator), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CommandInhibitFlag (C enumerator), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CommandLineLevelFlag (C enumerator), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CommandTimeoutFlag (C enumerator), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CommandTypeAbortFlag (C enumerator), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CommandTypeResumeFlag (C enumerator), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_CommandTypeSuspendFlag (C enumerator), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Data0LineLevelFlag (C enumerator), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Data1LineLevelFlag (C enumerator), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Data2LineLevelFlag (C enumerator), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Data3LineLevelFlag (C enumerator), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Data4LineLevelFlag (C enumerator), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Data5LineLevelFlag (C enumerator), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Data6LineLevelFlag (C enumerator), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Data7LineLevelFlag (C enumerator), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DataCompleteFlag (C enumerator), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DataCrcErrorFlag (C enumerator), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DataDMAFlag (C enumerator), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DataEndBitErrorFlag (C enumerator), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DataErrorFlag (C enumerator), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DataFlag (C enumerator), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DataInhibitFlag (C enumerator), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DataLineActiveFlag (C enumerator), 952

drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DataPresentFlag (*C enumerator*), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DataReadFlag (*C enumerator*), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DataTimeoutFlag (*C enumerator*), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DelaySettingFinishedFlag (*C enumerator*), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DmaCompleteFlag (*C enumerator*), 953
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_DmaErrorFlag (*C enumerator*), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_EnableAutoCommand12Flag (*C enumerator*), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_EnableAutoCommand23Flag (*C enumerator*), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_EnableBlockCountFlag (*C enumerator*), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_EnableCrcCheckFlag (*C enumerator*), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_EnableDmaFlag (*C enumerator*), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_EnableIndexCheckFlag (*C enumerator*), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ErrorFlag (*C enumerator*), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ExactBlockNumberReadFlag (*C enumerator*), 959
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ExecuteTuning (*C enumerator*), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12CrcError (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12Error (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12IndexError (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12NotExecuted (*C enumerator*), 956
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12NotIssued (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12Timeout (*C enumerator*), 956
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventCardInt (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventCommandCrcError (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventCommandEndBitError (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventCommandIndexError (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventCommandTimeout (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventDataCrcError (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventDataEndBitError (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventDataTimeout (*C enumerator*), 957

drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventDmaError (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventEndBitError (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventsAll (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ForceEventTuningError (*C enumerator*), 957
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_InterruptAtBlockGapFlag (*C enumerator*), 959
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_MultipleBlockFlag (*C enumerator*), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ReadDoneNo8CLK (*C enumerator*), 959
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ReadTransferActiveFlag (*C enumerator*), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ReadWaitControlFlag (*C enumerator*), 959
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ResetAll (*C enumerator*), 950
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ResetCommand (*C enumerator*), 950
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ResetData (*C enumerator*), 950
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ResetsAll (*C enumerator*), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ResetTuning (*C enumerator*), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ResponseLength48BusyFlag (*C enumerator*), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ResponseLength48Flag (*C enumerator*), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ResponseLength136Flag (*C enumerator*), 951
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ReTuningEventFlag (*C enumerator*), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_ReTuningRequestFlag (*C enumerator*), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_SdClockStableFlag (*C enumerator*), 952
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_SDR104TuningFlag (*C enumerator*), 955
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_StopAtBlockGapFlag (*C enumerator*), 959
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Support4BitFlag (*C enumerator*), 950
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_Support8BitFlag (*C enumerator*), 950
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_SupportAdmaFlag (*C enumerator*), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_SupportDDR50Flag (*C enumerator*), 950
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_SupportDmaFlag (*C enumerator*), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_SupportHighSpeedFlag (*C enumerator*), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_SupportSDR50Flag (*C enumerator*), 950
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_SupportSDR104Flag (*C enumerator*), 950

drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_SupportSuspendResumeFlag (*C enumerator*), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_SupportV300Flag (*C enumerator*), 950
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_SupportV330Flag (*C enumerator*), 949
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_TransferDataBoot (*C enumerator*), 962
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_TransferDataBootcontinous (*C enumerator*), 962
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_TransferDataNormal (*C enumerator*), 962
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_TransferDataTuning (*C enumerator*), 962
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_TuningErrorFlag (*C enumerator*), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_TuningPassFlag (*C enumerator*), 954
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_TuningSampleClockSel (*C enumerator*), 956
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_WakeupEventOnCardInsert (*C enumerator*), 950
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_WakeupEventOnCardInt (*C enumerator*), 950
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_WakeupEventOnCardRemove (*C enumerator*), 950
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_WakeupEventsAll (*C enumerator*), 950
drivers_RT_RT700_MIMXRT798S.PhonyNameDueToError.kUSDHC_WriteTransferActiveFlag (*C enumerator*), 952
drivers_RT_RT700_MIMXRT798S.pin (*C var*), 516
drivers_RT_RT700_MIMXRT798S.PININT_BITSLICE_CFG_MASK (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.PININT_BITSLICE_CFG_START (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.PININT_BITSLICE_ENDP_MASK (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.PININT_BITSLICE_SRC_MASK (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.PININT_BITSLICE_SRC_START (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.pint_cb_t (*C type*), 778
drivers_RT_RT700_MIMXRT798S.PINT_CLOCKS (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.PINT_Deinit (*C function*), 783
drivers_RT_RT700_MIMXRT798S.PINT_DisableCallback (*C function*), 783
drivers_RT_RT700_MIMXRT798S.PINT_DisableCallbackByIndex (*C function*), 784
drivers_RT_RT700_MIMXRT798S.PINT_DisableInterruptByIndex (*C function*), 784
drivers_RT_RT700_MIMXRT798S.PINT_DriverIRQHandler (*C function*), 784
drivers_RT_RT700_MIMXRT798S.PINT_EnableCallback (*C function*), 783
drivers_RT_RT700_MIMXRT798S.PINT_EnableCallbackByIndex (*C function*), 784
drivers_RT_RT700_MIMXRT798S.PINT_EnableInterruptByIndex (*C function*), 784
drivers_RT_RT700_MIMXRT798S.PINT_Init (*C function*), 778
drivers_RT_RT700_MIMXRT798S.PINT_PatternMatchConfig (*C function*), 781
drivers_RT_RT700_MIMXRT798S.PINT_PatternMatchDisable (*C function*), 782
drivers_RT_RT700_MIMXRT798S.PINT_PatternMatchDisableRXEV (*C function*), 783
drivers_RT_RT700_MIMXRT798S.PINT_PatternMatchEnable (*C function*), 782
drivers_RT_RT700_MIMXRT798S.PINT_PatternMatchEnableRXEV (*C function*), 783
drivers_RT_RT700_MIMXRT798S.PINT_PatternMatchGetConfig (*C function*), 781
drivers_RT_RT700_MIMXRT798S.PINT_PatternMatchGetStatus (*C function*), 782
drivers_RT_RT700_MIMXRT798S.PINT_PatternMatchGetStatusAll (*C function*), 782
drivers_RT_RT700_MIMXRT798S.PINT_PatternMatchResetDetectLogic (*C function*), 782
drivers_RT_RT700_MIMXRT798S.PINT_PIN_BOTH_EDGE (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.pint_pin_enable_t (*C type*), 778

drivers_RT_RT700_MIMXRT798S.PINT_PIN_FALL_EDGE (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.PINT_PIN_HIGH_LEVEL (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.PINT_PIN_INT_EDGE (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.PINT_PIN_INT_FALL_OR_HIGH_LEVEL (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.PINT_PIN_INT_LEVEL (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.PINT_PIN_INT_RISE (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.pint_pin_int_t (*C type*), 778
drivers_RT_RT700_MIMXRT798S.PINT_PIN_LOW_LEVEL (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.PINT_PIN_RISE_EDGE (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptClrFallFlag (*C function*), 780
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptClrFallFlagAll (*C function*), 780
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptClrRiseFlag (*C function*), 780
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptClrRiseFlagAll (*C function*), 781
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptClrStatus (*C function*), 779
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptClrStatusAll (*C function*), 779
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptConfig (*C function*), 779
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptGetConfig (*C function*), 779
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptGetFallFlag (*C function*), 780
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptGetFallFlagAll (*C function*), 780
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptGetRiseFlag (*C function*), 781
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptGetRiseFlagAll (*C function*), 781
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptGetStatus (*C function*), 779
drivers_RT_RT700_MIMXRT798S.PINT_PinInterruptGetStatusAll (*C function*), 780
drivers_RT_RT700_MIMXRT798S.pint_pmatch_bslice_cfg_t (*C type*), 778
drivers_RT_RT700_MIMXRT798S.pint_pmatch_bslice_t (*C type*), 778
drivers_RT_RT700_MIMXRT798S.pint_pmatch_cfg_t (*C type*), 778
drivers_RT_RT700_MIMXRT798S.pint_pmatch_input_src_t (*C type*), 778
drivers_RT_RT700_MIMXRT798S.PINT_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.PINT_SetCallback (*C function*), 778
drivers_RT_RT700_MIMXRT798S.pint_status_t (*C type*), 778
drivers_RT_RT700_MIMXRT798S.PINT_USE_LEGACY_CALLBACK (*C macro*), 785
drivers_RT_RT700_MIMXRT798S.PINTSEL_PMUX_ID (*C macro*), 512
drivers_RT_RT700_MIMXRT798S.pixelDepth (*C var*), 537
drivers_RT_RT700_MIMXRT798S.pixelFormat (*C var*), 537
drivers_RT_RT700_MIMXRT798S.pLdrFnc_v3_t (*C type*), 880
drivers_RT_RT700_MIMXRT798S.PMC_PDRCFG0 (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.PMC_PDRCFG1 (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.PMC_PDRCFG2 (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.PMC_PDRCFG3 (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.PMC_PDRCFG4 (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.PMC_PDRCFG5 (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.pmic_mode_reg_t (*C type*), 806
drivers_RT_RT700_MIMXRT798S.PMUX_SHIFT (*C macro*), 512
drivers_RT_RT700_MIMXRT798S.PNGDEC_ClearStatusFlags (*C function*), 788
drivers_RT_RT700_MIMXRT798S.PNGDEC_CLOCKS (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.pngdec_color_type_t (*C type*), 787
drivers_RT_RT700_MIMXRT798S.pngdec_config_t (*C type*), 787
drivers_RT_RT700_MIMXRT798S.PNGDEC_Decode (*C function*), 790
drivers_RT_RT700_MIMXRT798S.PNGDEC_Deinit (*C function*), 788
drivers_RT_RT700_MIMXRT798S.PNGDEC_DisableInterrupts (*C function*), 788
drivers_RT_RT700_MIMXRT798S.PNGDEC_Enable (*C function*), 789
drivers_RT_RT700_MIMXRT798S.PNGDEC_EnableAncillary (*C function*), 789
drivers_RT_RT700_MIMXRT798S.PNGDEC_EnableInterrupts (*C function*), 788
drivers_RT_RT700_MIMXRT798S.pngdec_endian_mode_t (*C type*), 787
drivers_RT_RT700_MIMXRT798S.PNGDEC_GetDefaultConfig (*C function*), 788
drivers_RT_RT700_MIMXRT798S.PNGDEC_GetStatusFlags (*C function*), 788
drivers_RT_RT700_MIMXRT798S.pngdec_image_t (*C type*), 787
drivers_RT_RT700_MIMXRT798S.PNGDEC_Init (*C function*), 788

drivers_RT_RT700_MIMXRT798S.PNGDEC_ParseHeader (*C function*), 789
drivers_RT_RT700_MIMXRT798S.PNGDEC_Reset (*C function*), 789
drivers_RT_RT700_MIMXRT798S.PNGDEC_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.PNGDEC_SetOutputBuffer (*C function*), 789
drivers_RT_RT700_MIMXRT798S.PNGDEC_SetPngBuffer (*C function*), 789
drivers_RT_RT700_MIMXRT798S.PNGDEC_StartDecode (*C function*), 789
drivers_RT_RT700_MIMXRT798S.port (*C var*), 516
drivers_RT_RT700_MIMXRT798S.POWER_ApplyPD (*C function*), 808
drivers_RT_RT700_MIMXRT798S.POWER_CalcVoltLevel (*C function*), 811
drivers_RT_RT700_MIMXRT798S.POWER_ClearEventFlags (*C function*), 808
drivers_RT_RT700_MIMXRT798S.POWER_ConfigLvdSetpoints (*C function*), 812
drivers_RT_RT700_MIMXRT798S.POWER_ConfigRegulatorSetpoints (*C function*), 811
drivers_RT_RT700_MIMXRT798S.POWER_ConfigRegulatorSetpointsForFreq (*C function*), 811
drivers_RT_RT700_MIMXRT798S.POWER_DEFAULT_LVD_VOLT (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.POWER_DEFAULT_PMICMODE_DPD (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.POWER_DEFAULT_PMICMODE_DS (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.POWER_DEFAULT_PMICMODE_DSR (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.POWER_DEFAULT_PMICMODE_FDPD (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.POWER_DisableDeepSleep (*C function*), 808
drivers_RT_RT700_MIMXRT798S.POWER_DisableDMAHWWake (*C function*), 809
drivers_RT_RT700_MIMXRT798S.POWER_DisableInterrupts (*C function*), 808
drivers_RT_RT700_MIMXRT798S.POWER_DisableLPRequestMask (*C function*), 810
drivers_RT_RT700_MIMXRT798S.POWER_DisablePD (*C function*), 807
drivers_RT_RT700_MIMXRT798S.POWER_DisableResets (*C function*), 808
drivers_RT_RT700_MIMXRT798S.POWER_EnableAutoWake (*C function*), 809
drivers_RT_RT700_MIMXRT798S.POWER_EnableDeepSleep (*C function*), 808
drivers_RT_RT700_MIMXRT798S.POWER_EnableDMAHWWake (*C function*), 809
drivers_RT_RT700_MIMXRT798S.POWER_EnableInterrupts (*C function*), 808
drivers_RT_RT700_MIMXRT798S.POWER_EnableLPRequestMask (*C function*), 809
drivers_RT_RT700_MIMXRT798S.POWER_EnablePD (*C function*), 807
drivers_RT_RT700_MIMXRT798S.POWER_EnableResets (*C function*), 808
drivers_RT_RT700_MIMXRT798S.POWER_EnableRunAFBB (*C function*), 810
drivers_RT_RT700_MIMXRT798S.POWER_EnableRunNBB (*C function*), 810
drivers_RT_RT700_MIMXRT798S.POWER_EnableRunRBB (*C function*), 810
drivers_RT_RT700_MIMXRT798S.POWER_EnableSleepAFBB (*C function*), 810
drivers_RT_RT700_MIMXRT798S.POWER_EnableSleepNBB (*C function*), 811
drivers_RT_RT700_MIMXRT798S.POWER_EnableSleepRBB (*C function*), 810
drivers_RT_RT700_MIMXRT798S.POWER_EnterDeepSleep (*C function*), 814
drivers_RT_RT700_MIMXRT798S.POWER_EnterPowerMode (*C function*), 815
drivers_RT_RT700_MIMXRT798S.POWER_EnterSleep (*C function*), 814
drivers_RT_RT700_MIMXRT798S.POWER_GetEventFlags (*C function*), 808
drivers_RT_RT700_MIMXRT798S.POWER_GetLibVersion (*C function*), 815
drivers_RT_RT700_MIMXRT798S.POWER_GetLvdSetpoints (*C function*), 813
drivers_RT_RT700_MIMXRT798S.POWER_GetPmicMode (*C function*), 808
drivers_RT_RT700_MIMXRT798S.POWER_IOBankClearIsolationHold (*C function*), 814
drivers_RT_RT700_MIMXRT798S.POWER_IOBankIsolationHold (*C function*), 814
drivers_RT_RT700_MIMXRT798S.POWER_LDO_SAFE_MARGIN (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.POWER_LP_REQ_CTRL_REG_OFFSET (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.POWER_LP_REQ_STATE_BIT_OFFSET (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.POWER_LP_REQ_STATE_REG_OFFSET (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.power_lvd_voltage_t (*C type*), 807
drivers_RT_RT700_MIMXRT798S.POWER_MAKE_LP_BITS (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.power_mode_cfg_t (*C type*), 806
drivers_RT_RT700_MIMXRT798S.POWER_ModuleEnterLPRequest (*C function*), 807
drivers_RT_RT700_MIMXRT798S.POWER_ModuleExitLPRequest (*C function*), 807
drivers_RT_RT700_MIMXRT798S.power_pad_vrange_val_t (*C type*), 806
drivers_RT_RT700_MIMXRT798S.power_regulator_t (*C type*), 807
drivers_RT_RT700_MIMXRT798S.power_regulator_voltage_t (*C type*), 807

drivers_RT_RT700_MIMXRT798S.POWER_RequestDeepPowerDown (*C function*), 815
drivers_RT_RT700_MIMXRT798S.POWER_RequestDSR (*C function*), 815
drivers_RT_RT700_MIMXRT798S.POWER_RequestFullDeepPowerDown (*C function*), 815
drivers_RT_RT700_MIMXRT798S.POWER_ResetIOBank (*C function*), 814
drivers_RT_RT700_MIMXRT798S.POWER_RETRY_TIMES (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.POWER_SelectRunSetpoint (*C function*), 813
drivers_RT_RT700_MIMXRT798S.POWER_SelectSleepSetpoint (*C function*), 813
drivers_RT_RT700_MIMXRT798S.POWER_SetPmicMode (*C function*), 807
drivers_RT_RT700_MIMXRT798S.POWER_SetRunRegulatorMode (*C function*), 813
drivers_RT_RT700_MIMXRT798S.POWER_SetSleepRegulatorMode (*C function*), 814
drivers_RT_RT700_MIMXRT798S.power_vdd_src_t (*C type*), 807
drivers_RT_RT700_MIMXRT798S.PRIVATE_CCTRL0_OFFSET (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.PRIVATE_CSTAT0_OFFSET (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.PRIVATE_TCTRL0_OFFSET (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.PRIVATE_TCTRL1_OFFSET (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.PRIVATE_TCTRL2_OFFSET (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.PRIVATE_TSTAT0_OFFSET (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.PRIVATE_TSTAT1_OFFSET (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.PUF_ACTIVATION_CODE_SIZE (*C macro*), 826
drivers_RT_RT700_MIMXRT798S.PUF_BlockCommand (*C function*), 824
drivers_RT_RT700_MIMXRT798S.puf_config_t (*C struct*), 826
drivers_RT_RT700_MIMXRT798S.PUF_Deinit (*C function*), 821
drivers_RT_RT700_MIMXRT798S.puf_endianness_t (*C type*), 821
drivers_RT_RT700_MIMXRT798S.PUF_Enroll (*C function*), 822
drivers_RT_RT700_MIMXRT798S.PUF_GenerateRandom (*C function*), 824
drivers_RT_RT700_MIMXRT798S.PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE (*C macro*), 826
drivers_RT_RT700_MIMXRT798S.PUF_GetDefaultConfig (*C function*), 821
drivers_RT_RT700_MIMXRT798S.PUF_GetKey (*C function*), 822
drivers_RT_RT700_MIMXRT798S.PUF_Init (*C function*), 821
drivers_RT_RT700_MIMXRT798S.puf_key_ctx_t (*C struct*), 826
drivers_RT_RT700_MIMXRT798S.puf_key_dest_t (*C type*), 821
drivers_RT_RT700_MIMXRT798S.puf_key_scope_t (*C type*), 821
drivers_RT_RT700_MIMXRT798S.puf_result_code_t (*C type*), 821
drivers_RT_RT700_MIMXRT798S.puf_sec_level_t (*C type*), 821
drivers_RT_RT700_MIMXRT798S.PUF_SetCtxMask (*C function*), 825
drivers_RT_RT700_MIMXRT798S.PUF_SetLock (*C function*), 824
drivers_RT_RT700_MIMXRT798S.PUF_Start (*C function*), 822
drivers_RT_RT700_MIMXRT798S.PUF_Stop (*C function*), 822
drivers_RT_RT700_MIMXRT798S.PUF_Test (*C function*), 824
drivers_RT_RT700_MIMXRT798S.PUF_Unwrap (*C function*), 823
drivers_RT_RT700_MIMXRT798S.PUF_Wrap (*C function*), 823
drivers_RT_RT700_MIMXRT798S.PUF_WrapGeneratedRandom (*C function*), 823
drivers_RT_RT700_MIMXRT798S.PUF_Zeroize (*C function*), 824
drivers_RT_RT700_MIMXRT798S.PVTS_ClearAlertCount (*C function*), 828
drivers_RT_RT700_MIMXRT798S.PVTS_Deinit (*C function*), 827
drivers_RT_RT700_MIMXRT798S.PVTS_DELAY_MAX (*C macro*), 828
drivers_RT_RT700_MIMXRT798S.PVTS_DELAY_MIN (*C macro*), 828
drivers_RT_RT700_MIMXRT798S.pvts_delay_t (*C type*), 826
drivers_RT_RT700_MIMXRT798S.PVTS_DisableAlertCount (*C function*), 828
drivers_RT_RT700_MIMXRT798S.PVTS_EnableAlertCount (*C function*), 828
drivers_RT_RT700_MIMXRT798S.PVTS_GET_A_PARAMETER_FROM_FUSE_VALUE (*C macro*), 828
drivers_RT_RT700_MIMXRT798S.PVTS_GET_B_PARAMETER_FROM_FUSE_VALUE (*C macro*), 828
drivers_RT_RT700_MIMXRT798S.PVTS_GET_DELAY0_FROM_FUSE_VALUE (*C macro*), 828
drivers_RT_RT700_MIMXRT798S.PVTS_GET_DELAY1_FROM_FUSE_VALUE (*C macro*), 828
drivers_RT_RT700_MIMXRT798S.PVTS_GetAlertCount (*C function*), 828

drivers_RT_RT700_MIMXRT798S.PVTS_GetDelay (*C function*), 827
drivers_RT_RT700_MIMXRT798S.PVTS_Init (*C function*), 827
drivers_RT_RT700_MIMXRT798S.pvts_sensor (*C enum*), 826
drivers_RT_RT700_MIMXRT798S.pvts_sensor_t (*C type*), 826
drivers_RT_RT700_MIMXRT798S.pvts_sensor.kPVTS_Sensor0 (*C enumerator*), 826
drivers_RT_RT700_MIMXRT798S.pvts_sensor.kPVTS_Sensor1 (*C enumerator*), 826
drivers_RT_RT700_MIMXRT798S.PVTS_SetDelay (*C function*), 827
drivers_RT_RT700_MIMXRT798S.PVTS_Start (*C function*), 827
drivers_RT_RT700_MIMXRT798S.PVTS_Stop (*C function*), 827
drivers_RT_RT700_MIMXRT798S.RAMFUNCTION_SECTION_CODE (*C macro*), 544
drivers_RT_RT700_MIMXRT798S.range (*C var*), 266
drivers_RT_RT700_MIMXRT798S.range_header (*C struct*), 882
drivers_RT_RT700_MIMXRT798S.range_header_expansion (*C struct*), 882
drivers_RT_RT700_MIMXRT798S.recovery_boot_cfg0 (*C var*), 841
drivers_RT_RT700_MIMXRT798S.recovery_boot_cfg1 (*C var*), 841
drivers_RT_RT700_MIMXRT798S.refDiv (*C var*), 266
drivers_RT_RT700_MIMXRT798S.reserved0 (*C var*), 841
drivers_RT_RT700_MIMXRT798S.reserved1 (*C var*), 840, 841
drivers_RT_RT700_MIMXRT798S.RESET_ClearPeripheralReset (*C function*), 838
drivers_RT_RT700_MIMXRT798S.RESET_ClearResetSources (*C function*), 838
drivers_RT_RT700_MIMXRT798S.RESET_GetDomainResetStatus (*C function*), 838
drivers_RT_RT700_MIMXRT798S.RESET_GetPreviousResetSources (*C function*), 838
drivers_RT_RT700_MIMXRT798S.reset_ip_name_t (*C type*), 838
drivers_RT_RT700_MIMXRT798S.RESET_PeripheralReset (*C function*), 838
drivers_RT_RT700_MIMXRT798S.RESET_ReleasePeripheralReset (*C function*), 838
drivers_RT_RT700_MIMXRT798S.RESET_SetPeripheralReset (*C function*), 838
drivers_RT_RT700_MIMXRT798S.reseverd0 (*C var*), 840
drivers_RT_RT700_MIMXRT798S.restore_sequence (*C var*), 840
drivers_RT_RT700_MIMXRT798S.RST_CTL0_PSCCTL0 (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.RST_CTL0_PSCCTL1 (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.RST_CTL0_PSCCTL2 (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.RST_CTL0_PSCCTL3 (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.RST_CTL0_PSCCTL4 (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.RST_CTL0_PSCCTL5 (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.RST_CTL1_PSCCTL0 (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.RST_CTL2_PSCCTL0 (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.RST_CTL3_PSCCTL0 (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.RST_CTL3_PSCCTL1 (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.RST_CTL4_PSCCTL0 (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.RST_CTL4_PSCCTL1 (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.rstctl_reset_source_t (*C type*), 837
drivers_RT_RT700_MIMXRT798S.RSTCTL_RSTn_t (*C type*), 837
drivers_RT_RT700_MIMXRT798S.RTC_CLOCKS (*C macro*), 260
drivers_RT_RT700_MIMXRT798S.rxDmaHandle (*C var*), 1050
drivers_RT_RT700_MIMXRT798S.s_flexioBases (*C var*), 350
drivers_RT_RT700_MIMXRT798S.s_flexioClocks (*C var*), 350
drivers_RT_RT700_MIMXRT798S.s_lpuartIRQ (*C var*), 691
drivers_RT_RT700_MIMXRT798S.SAI3CLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.SAI3FCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.sai_bclk_source_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.sai_bit_clock_t (*C type*), 866
drivers_RT_RT700_MIMXRT798S.sai_clock_polarity_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.SAI_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.sai_config_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.sai_data_order_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.sai_data_pin_state_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.SAI_Deinit (*C function*), 842
drivers_RT_RT700_MIMXRT798S.SAI_DriverIRQHandler (*C function*), 857

drivers_RT_RT700_MIMXRT798S.sai_edma_callback_t (*C type*), 877
drivers_RT_RT700_MIMXRT798S.sai_edma_handle (*C struct*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle_t (*C type*), 877
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.bytesPerFrame (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.callback (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.channel (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.channelMask (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.channelNums (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.count (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.dmaHandle (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.interleaveType (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.nbytes (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.queueDriver (*C var*), 879
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.queueUser (*C var*), 879
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.saiQueue (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.state (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.tcd (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.transferSize (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_handle.userData (*C var*), 878
drivers_RT_RT700_MIMXRT798S.sai_edma_interleave_t (*C type*), 877
drivers_RT_RT700_MIMXRT798S.sai_fifo_combine_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.sai_fifo_packing_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.sai_fifo_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.sai_frame_sync_len_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.sai_frame_sync_t (*C type*), 866
drivers_RT_RT700_MIMXRT798S.SAI_GetClassicI2SConfig (*C function*), 846
drivers_RT_RT700_MIMXRT798S.SAI_GetDSPConfig (*C function*), 847
drivers_RT_RT700_MIMXRT798S.SAI_GetLeftJustifiedConfig (*C function*), 846
drivers_RT_RT700_MIMXRT798S.SAI_GetRightJustifiedConfig (*C function*), 846
drivers_RT_RT700_MIMXRT798S.SAI_GetTDMConfig (*C function*), 846
drivers_RT_RT700_MIMXRT798S.sai_handle_t (*C type*), 866
drivers_RT_RT700_MIMXRT798S.SAI_Init (*C function*), 842
drivers_RT_RT700_MIMXRT798S.sai_master_clock_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.sai_master_slave_t (*C type*), 864
drivers_RT_RT700_MIMXRT798S.sai_mono_stereo_t (*C type*), 864
drivers_RT_RT700_MIMXRT798S.sai_protocol_t (*C type*), 864
drivers_RT_RT700_MIMXRT798S.SAI_ReadBlocking (*C function*), 853
drivers_RT_RT700_MIMXRT798S.SAI_ReadData (*C function*), 853
drivers_RT_RT700_MIMXRT798S.SAI_ReadMultiChannelBlocking (*C function*), 853
drivers_RT_RT700_MIMXRT798S.sai_reset_type_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.SAI_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.SAI_RxClearStatusFlags (*C function*), 848
drivers_RT_RT700_MIMXRT798S.SAI_RxDisableInterrupts (*C function*), 851
drivers_RT_RT700_MIMXRT798S.SAI_RxEnable (*C function*), 843
drivers_RT_RT700_MIMXRT798S.SAI_RxEnableDMA (*C function*), 851
drivers_RT_RT700_MIMXRT798S.SAI_RxEnableInterrupts (*C function*), 850
drivers_RT_RT700_MIMXRT798S.SAI_RxGetDataRegisterAddress (*C function*), 852
drivers_RT_RT700_MIMXRT798S.SAI_RxGetStatusFlag (*C function*), 847
drivers_RT_RT700_MIMXRT798S.SAI_RxReset (*C function*), 843
drivers_RT_RT700_MIMXRT798S.SAI_RxSetBitclockConfig (*C function*), 844
drivers_RT_RT700_MIMXRT798S.SAI_RxSetBitClockDirection (*C function*), 843
drivers_RT_RT700_MIMXRT798S.SAI_RxSetBitClockPolarity (*C function*), 849
drivers_RT_RT700_MIMXRT798S.SAI_RxSetBitClockRate (*C function*), 844
drivers_RT_RT700_MIMXRT798S.SAI_RxSetChannelFIFOMask (*C function*), 848
drivers_RT_RT700_MIMXRT798S.SAI_RxSetConfig (*C function*), 846
drivers_RT_RT700_MIMXRT798S.SAI_RxSetDataOrder (*C function*), 849
drivers_RT_RT700_MIMXRT798S.SAI_RxSetFifoConfig (*C function*), 845
drivers_RT_RT700_MIMXRT798S.SAI_RxSetFIFOErrorContinue (*C function*), 850

drivers_RT_RT700_MIMXRT798S.SAI_RxSetFIFOPacking (*C function*), 849
drivers_RT_RT700_MIMXRT798S.SAI_RxSetFrameSyncConfig (*C function*), 845
drivers_RT_RT700_MIMXRT798S.SAI_RxSetFrameSyncDirection (*C function*), 843
drivers_RT_RT700_MIMXRT798S.SAI_RxSetFrameSyncPolarity (*C function*), 849
drivers_RT_RT700_MIMXRT798S.SAI_RxSetSerialDataConfig (*C function*), 845
drivers_RT_RT700_MIMXRT798S.SAI_RxSoftwareReset (*C function*), 848
drivers_RT_RT700_MIMXRT798S.sai_sample_rate_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.sai_serial_data_t (*C type*), 866
drivers_RT_RT700_MIMXRT798S.SAI_SetMasterClockConfig (*C function*), 844
drivers_RT_RT700_MIMXRT798S.sai_sync_mode_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.sai_transceiver_t (*C type*), 866
drivers_RT_RT700_MIMXRT798S.sai_transceiver_type_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.sai_transfer_callback_t (*C type*), 866
drivers_RT_RT700_MIMXRT798S.sai_transfer_format_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.sai_transfer_t (*C type*), 866
drivers_RT_RT700_MIMXRT798S.SAI_TransferAbortReceive (*C function*), 856
drivers_RT_RT700_MIMXRT798S.SAI_TransferAbortReceiveEDMA (*C function*), 876
drivers_RT_RT700_MIMXRT798S.SAI_TransferAbortSend (*C function*), 856
drivers_RT_RT700_MIMXRT798S.SAI_TransferAbortSendEDMA (*C function*), 876
drivers_RT_RT700_MIMXRT798S.SAI_TransferGetReceiveCount (*C function*), 855
drivers_RT_RT700_MIMXRT798S.SAI_TransferGetReceiveCountEDMA (*C function*), 876
drivers_RT_RT700_MIMXRT798S.SAI_TransferGetSendCount (*C function*), 855
drivers_RT_RT700_MIMXRT798S.SAI_TransferGetSendCountEDMA (*C function*), 876
drivers_RT_RT700_MIMXRT798S.SAI_TransferGetValidTransferSlotsEDMA (*C function*), 877
drivers_RT_RT700_MIMXRT798S.SAI_TransferReceiveEDMA (*C function*), 874
drivers_RT_RT700_MIMXRT798S.SAI_TransferReceiveLoopEDMA (*C function*), 875
drivers_RT_RT700_MIMXRT798S.SAI_TransferReceiveNonBlocking (*C function*), 855
drivers_RT_RT700_MIMXRT798S.SAI_TransferRxCreateHandle (*C function*), 854
drivers_RT_RT700_MIMXRT798S.SAI_TransferRxCreateHandleEDMA (*C function*), 872
drivers_RT_RT700_MIMXRT798S.SAI_TransferRxHandleIRQ (*C function*), 857
drivers_RT_RT700_MIMXRT798S.SAI_TransferRxSetConfig (*C function*), 854
drivers_RT_RT700_MIMXRT798S.SAI_TransferRxSetConfigEDMA (*C function*), 873
drivers_RT_RT700_MIMXRT798S.SAI_TransferSendEDMA (*C function*), 873
drivers_RT_RT700_MIMXRT798S.SAI_TransferSendLoopEDMA (*C function*), 874
drivers_RT_RT700_MIMXRT798S.SAI_TransferSendNonBlocking (*C function*), 854
drivers_RT_RT700_MIMXRT798S.SAI_TransferSetInterleaveType (*C function*), 872
drivers_RT_RT700_MIMXRT798S.SAI_TransferTerminateReceive (*C function*), 856
drivers_RT_RT700_MIMXRT798S.SAI_TransferTerminateReceiveEDMA (*C function*), 876
drivers_RT_RT700_MIMXRT798S.SAI_TransferTerminateSend (*C function*), 856
drivers_RT_RT700_MIMXRT798S.SAI_TransferTerminateSendEDMA (*C function*), 875
drivers_RT_RT700_MIMXRT798S.SAI_TransferTxCreateHandle (*C function*), 853
drivers_RT_RT700_MIMXRT798S.SAI_TransferTxCreateHandleEDMA (*C function*), 872
drivers_RT_RT700_MIMXRT798S.SAI_TransferTxHandleIRQ (*C function*), 857
drivers_RT_RT700_MIMXRT798S.SAI_TransferTxSetConfig (*C function*), 854
drivers_RT_RT700_MIMXRT798S.SAI_TransferTxSetConfigEDMA (*C function*), 873
drivers_RT_RT700_MIMXRT798S.SAI_TxClearStatusFlags (*C function*), 847
drivers_RT_RT700_MIMXRT798S.SAI_TxDisableInterrupts (*C function*), 850
drivers_RT_RT700_MIMXRT798S.SAI_TxEnable (*C function*), 843
drivers_RT_RT700_MIMXRT798S.SAI_TxEnableDMA (*C function*), 851
drivers_RT_RT700_MIMXRT798S.SAI_TxEnableInterrupts (*C function*), 850
drivers_RT_RT700_MIMXRT798S.SAI_TxGetDataRegisterAddress (*C function*), 851
drivers_RT_RT700_MIMXRT798S.SAI_TxGetStatusFlag (*C function*), 847
drivers_RT_RT700_MIMXRT798S.SAI_TxReset (*C function*), 843
drivers_RT_RT700_MIMXRT798S.SAI_TxSetBitclockConfig (*C function*), 844
drivers_RT_RT700_MIMXRT798S.SAI_TxSetBitClockDirection (*C function*), 843
drivers_RT_RT700_MIMXRT798S.SAI_TxSetBitClockPolarity (*C function*), 849
drivers_RT_RT700_MIMXRT798S.SAI_TxSetBitClockRate (*C function*), 844
drivers_RT_RT700_MIMXRT798S.SAI_TxSetChannelFIFOMask (*C function*), 848

drivers_RT_RT700_MIMXRT798S.SAI_TxSetConfig (*C function*), 845
drivers_RT_RT700_MIMXRT798S.SAI_TxSetDataOrder (*C function*), 849
drivers_RT_RT700_MIMXRT798S.SAI_TxSetFifoConfig (*C function*), 845
drivers_RT_RT700_MIMXRT798S.SAI_TxSetFIFOErrorContinue (*C function*), 850
drivers_RT_RT700_MIMXRT798S.SAI_TxSetFIFOPacking (*C function*), 849
drivers_RT_RT700_MIMXRT798S.SAI_TxSetFrameSyncConfig (*C function*), 845
drivers_RT_RT700_MIMXRT798S.SAI_TxSetFrameSyncDirection (*C function*), 844
drivers_RT_RT700_MIMXRT798S.SAI_TxSetFrameSyncPolarity (*C function*), 849
drivers_RT_RT700_MIMXRT798S.SAI_TxSetSerialDataConfig (*C function*), 845
drivers_RT_RT700_MIMXRT798S.SAI_TxSoftwareReset (*C function*), 848
drivers_RT_RT700_MIMXRT798S.sai_word_width_t (*C type*), 865
drivers_RT_RT700_MIMXRT798S.SAI_WriteBlocking (*C function*), 852
drivers_RT_RT700_MIMXRT798S.SAI_WriteData (*C function*), 852
drivers_RT_RT700_MIMXRT798S.SAI_WriteMultiChannelBlocking (*C function*), 852
drivers_RT_RT700_MIMXRT798S.SAI_XFER_QUEUE_SIZE (*C macro*), 866
drivers_RT_RT700_MIMXRT798S.SARADCFCLKDIV_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.SARADCFCLKSEL_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.SB3_BYTES_PER_CHUNK (*C macro*), 881
drivers_RT_RT700_MIMXRT798S.sb3_cmd_t (*C type*), 881
drivers_RT_RT700_MIMXRT798S.sb3_config_memory_t (*C type*), 881
drivers_RT_RT700_MIMXRT798S.sb3_copy_memory_expansion_t (*C type*), 881
drivers_RT_RT700_MIMXRT798S.sb3_copy_memory_t (*C type*), 881
drivers_RT_RT700_MIMXRT798S.SB3_DATA_BUFFER_SIZE_IN_BYTE (*C macro*), 882
drivers_RT_RT700_MIMXRT798S.sb3_data_range_expansion_t (*C type*), 881
drivers_RT_RT700_MIMXRT798S.SB3_DATA_RANGE_HEADER_FLAGS_ERASE_MASK (*C macro*), 881
drivers_RT_RT700_MIMXRT798S.SB3_DATA_RANGE_HEADER_FLAGS_LOAD_MASK (*C macro*), 881
drivers_RT_RT700_MIMXRT798S.sb3_data_range_header_t (*C type*), 881
drivers_RT_RT700_MIMXRT798S.sb3_fill_memory_expansion_t (*C type*), 881
drivers_RT_RT700_MIMXRT798S.sb3_fill_memory_t (*C type*), 881
drivers_RT_RT700_MIMXRT798S.sb3_fw_ver_check_t (*C type*), 881
drivers_RT_RT700_MIMXRT798S.sb3_load_keyblob_t (*C type*), 881
drivers_RT_RT700_MIMXRT798S.sb3_section_header_t (*C type*), 881
drivers_RT_RT700_MIMXRT798S.SBLOADER_V3_CMD_SET_ALL (*C macro*), 881
drivers_RT_RT700_MIMXRT798S.SBLOADER_V3_CMD_SET_IN_ISP_MODE (*C macro*), 881
drivers_RT_RT700_MIMXRT798S.SBLOADER_V3_CMD_SET_IN_REC_MODE (*C macro*), 881
drivers_RT_RT700_MIMXRT798S.SC_ADD (*C macro*), 210
drivers_RT_RT700_MIMXRT798S.SC_ADD1 (*C macro*), 210
drivers_RT_RT700_MIMXRT798S.SC_ADD16 (*C macro*), 210
drivers_RT_RT700_MIMXRT798S.SC_ADD256 (*C macro*), 210
drivers_RT_RT700_MIMXRT798S.SC_CHECK (*C macro*), 210
drivers_RT_RT700_MIMXRT798S.SC_SUB (*C macro*), 210
drivers_RT_RT700_MIMXRT798S.SC_SUB1 (*C macro*), 210
drivers_RT_RT700_MIMXRT798S.SC_SUB16 (*C macro*), 210
drivers_RT_RT700_MIMXRT798S.SC_SUB256 (*C macro*), 210
drivers_RT_RT700_MIMXRT798S.SCT_CLOCKS (*C macro*), 258
drivers_RT_RT700_MIMXRT798S.SCT_EV_STATE_STATEMSKn (*C macro*), 906
drivers_RT_RT700_MIMXRT798S.SCT_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.SCTIMER_ClearEventInState (*C function*), 896
drivers_RT_RT700_MIMXRT798S.SCTIMER_ClearStatusFlags (*C function*), 890
drivers_RT_RT700_MIMXRT798S.sctimer_clock_mode_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.sctimer_clock_select_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.sctimer_complementary_pwm_handle_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.sctimer_config_t (*C type*), 906
drivers_RT_RT700_MIMXRT798S.sctimer_conflict_resolution_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.sctimer_counter_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.SCTIMER_CreateAndScheduleEvent (*C function*), 891

drivers_RT_RT700_MIMXRT798S.SCTIMER_Deinit (*C function*), 884
drivers_RT_RT700_MIMXRT798S.SCTIMER_DisableInterrupts (*C function*), 890
drivers_RT_RT700_MIMXRT798S.SCTIMER_DriverIRQHandler (*C function*), 897
drivers_RT_RT700_MIMXRT798S.SCTIMER_EnableInterrupts (*C function*), 889
drivers_RT_RT700_MIMXRT798S.sctimer_event_active_direction_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.sctimer_event_callback_t (*C type*), 906
drivers_RT_RT700_MIMXRT798S.sctimer_event_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.SCTIMER_EventHandleIRQ (*C function*), 897
drivers_RT_RT700_MIMXRT798S.SCTIMER_GetCaptureValue (*C function*), 897
drivers_RT_RT700_MIMXRT798S.SCTIMER_GetCounterState (*C function*), 892
drivers_RT_RT700_MIMXRT798S.SCTIMER_GetCOUNTValue (*C function*), 896
drivers_RT_RT700_MIMXRT798S.SCTIMER_GetCurrentState (*C function*), 891
drivers_RT_RT700_MIMXRT798S.SCTIMER_GetDefaultConfig (*C function*), 885
drivers_RT_RT700_MIMXRT798S.SCTIMER_GetEnabledInterrupts (*C function*), 890
drivers_RT_RT700_MIMXRT798S.SCTIMER_GetEventInState (*C function*), 896
drivers_RT_RT700_MIMXRT798S.SCTIMER_GetStatusFlags (*C function*), 890
drivers_RT_RT700_MIMXRT798S.SCTIMER_IncreaseState (*C function*), 891
drivers_RT_RT700_MIMXRT798S.SCTIMER_Init (*C function*), 884
drivers_RT_RT700_MIMXRT798S.sctimer_input_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.sctimer_interrupt_enable_t (*C type*), 906
drivers_RT_RT700_MIMXRT798S.sctimer_out_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.sctimer_pwm_level_select_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.sctimer_pwm_mode_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.sctimer_pwm_signal_param_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.sctimer_pwm_update_mode_t (*C type*), 905
drivers_RT_RT700_MIMXRT798S.SCTIMER_ScheduleEvent (*C function*), 891
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetCallback (*C function*), 892
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetCounterState (*C function*), 892
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetCOUNTValue (*C function*), 895
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetEventInState (*C function*), 896
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupCaptureAction (*C function*), 892
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupComplementaryPwm (*C function*), 888
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupCounterHaltAction (*C function*), 895
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupCounterLimitAction (*C function*), 894
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupCounterStartAction (*C function*), 895
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupCounterStopAction (*C function*), 895
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupDmaTriggerAction (*C function*), 895
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupEventActiveDirection (*C function*), 894
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupNextStateAction (*C function*), 893
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupNextStateActionwithLdMethod (*C function*), 893
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupOutputClearAction (*C function*), 894
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupOutputSetAction (*C function*), 894
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupOutputToggleAction (*C function*), 894
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupPwm (*C function*), 885
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupSharedPeriodPwm (*C function*), 886
drivers_RT_RT700_MIMXRT798S.SCTIMER_SetupStateLdMethodAction (*C function*), 893
drivers_RT_RT700_MIMXRT798S.SCTIMER_StartTimer (*C function*), 890
drivers_RT_RT700_MIMXRT798S.sctimer_status_flags_t (*C type*), 906
drivers_RT_RT700_MIMXRT798S.SCTIMER_StopTimer (*C function*), 890
drivers_RT_RT700_MIMXRT798S.SCTIMER_UpdateComplementaryPwmDutycycle (*C function*), 889
drivers_RT_RT700_MIMXRT798S.SCTIMER_UpdatePwmDutycycle (*C function*), 886
drivers_RT_RT700_MIMXRT798S.SCTIMER_UpdatePwmDutycycleByEvent (*C function*), 887
drivers_RT_RT700_MIMXRT798S.SDADC_ADC_CTL_2_SET_DITHER (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_ADC_CTL_2_SET_DITHER_MASK (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_ADC_CTL_2_SET_DITHER_SHIFT (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.sdadc_channel_config_t (*C type*), 920

drivers_RT_RT700_MIMXRT798S.sdadc_channel_group (*C type*), 920
drivers_RT_RT700_MIMXRT798S.sdadc_channel_mode_t (*C type*), 919
drivers_RT_RT700_MIMXRT798S.sdadc_channel_number_t (*C type*), 919
drivers_RT_RT700_MIMXRT798S.sdadc_channel_type_t (*C type*), 919
drivers_RT_RT700_MIMXRT798S.SDADC_CheckGlobalFifoInterrupted (*C function*), 913
drivers_RT_RT700_MIMXRT798S.SDADC_ClearGlobalFifoIntStatusFlag (*C function*), 913
drivers_RT_RT700_MIMXRT798S.SDADC_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.sdadc_config_t (*C type*), 920
drivers_RT_RT700_MIMXRT798S.SDADC_ConfigureChannelMode (*C function*), 910
drivers_RT_RT700_MIMXRT798S.SDADC_ConfigureDecimatorSampleRate (*C function*), 911
drivers_RT_RT700_MIMXRT798S.SDADC_ConfigureDecimatorVolume (*C function*), 911
drivers_RT_RT700_MIMXRT798S.SDADC_ConfigureDitherAmplitude (*C function*), 912
drivers_RT_RT700_MIMXRT798S.SDADC_ConfigureFifoWatermark (*C function*), 914
drivers_RT_RT700_MIMXRT798S.SDADC_ConfigureLdoDriveCapability (*C function*), 910
drivers_RT_RT700_MIMXRT798S.SDADC_ConfigureModulatorDcLoopReset (*C function*), 912
drivers_RT_RT700_MIMXRT798S.SDADC_ConfigureReferenceMode (*C function*), 909
drivers_RT_RT700_MIMXRT798S.SDADC_ControlDacCompensationEnable (*C function*), 910
drivers_RT_RT700_MIMXRT798S.SDADC_ControlDecimatorDcFilterEnable (*C function*), 910
drivers_RT_RT700_MIMXRT798S.SDADC_ControlDecimatorOutputInvertEnable (*C function*), 911
drivers_RT_RT700_MIMXRT798S.SDADC_ControlDecimatorReset (*C function*), 912
drivers_RT_RT700_MIMXRT798S.SDADC_ControlFifoIntEnable (*C function*), 913
drivers_RT_RT700_MIMXRT798S.SDADC_ControlFifoReadWriteReset (*C function*), 914
drivers_RT_RT700_MIMXRT798S.SDADC_ControlFifoWatermarkEnable (*C function*), 913
drivers_RT_RT700_MIMXRT798S.SDADC_ControlGlobalFifoIntEnable (*C function*), 913
drivers_RT_RT700_MIMXRT798S.SDADC_ControlLdoPowerEnable (*C function*), 909
drivers_RT_RT700_MIMXRT798S.SDADC_ControlModulatorDcLoopEnable (*C function*), 910
drivers_RT_RT700_MIMXRT798S.SDADC_ControlModulatorPowerEnable (*C function*), 909
drivers_RT_RT700_MIMXRT798S.SDADC_ControlModulatorReset (*C function*), 912
drivers_RT_RT700_MIMXRT798S.SDADC_ControlReferencePowerEnable (*C function*), 909
drivers_RT_RT700_MIMXRT798S.SDADC_CopyConvChannelFifoToBuffer (*C function*), 915
drivers_RT_RT700_MIMXRT798S.sdadc_decimator_sample_rate_t (*C type*), 919
drivers_RT_RT700_MIMXRT798S.SDADC_DECIMATOR_SET_SPEED_N (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_DECIMATOR_SET_SPEED_N_MASK (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_DECIMATOR_SET_SPEED_N_SHIFT (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_DECIMATOR_SET_SPEED_P (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_DECIMATOR_SET_SPEED_P_MASK (*C macro*), 920
drivers_RT_RT700_MIMXRT798S.SDADC_DECIMATOR_SET_SPEED_P_SHIFT (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_DECIMATOR_VOL_CTRL_N (*C macro*), 920
drivers_RT_RT700_MIMXRT798S.SDADC_DECIMATOR_VOL_CTRL_N_MASK (*C macro*), 920
drivers_RT_RT700_MIMXRT798S.SDADC_DECIMATOR_VOL_CTRL_N_SHIFT (*C macro*), 920
drivers_RT_RT700_MIMXRT798S.SDADC_DECIMATOR_VOL_CTRL_P (*C macro*), 920
drivers_RT_RT700_MIMXRT798S.SDADC_DECIMATOR_VOL_CTRL_P_MASK (*C macro*), 920
drivers_RT_RT700_MIMXRT798S.SDADC_DECIMATOR_VOL_CTRL_P_SHIFT (*C macro*), 920
drivers_RT_RT700_MIMXRT798S.SDADC_Deinit (*C function*), 908
drivers_RT_RT700_MIMXRT798S.SDADC_DoInitPowerUp (*C function*), 908
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_DEPTH (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_ENTRIES_AVAIL_MASK (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_FULL_INT_STATUS_MASK (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_FULL_INT_STATUS_SHIFT (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_MASK (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.sdadc_fifo_reset_operation_t (*C type*), 920
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_SHIFT (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_WATERMARK_CTL_N (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_WATERMARK_CTL_N_MASK (*C macro*), 921

drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_WATERMARK_CTL_N_SHIFT (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_WATERMARK_CTL_P (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_WATERMARK_CTL_P_MASK (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_WATERMARK_CTL_P_SHIFT (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_WATERMARK_STATUS_MASK (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_WATERMARK_STATUS_SHIFT (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_FIFO_WIDTH (*C macro*), 921
drivers_RT_RT700_MIMXRT798S.SDADC_GetConvChannelFifoAvailableEntries (*C function*), 914
drivers_RT_RT700_MIMXRT798S.SDADC_GetConvChannelFifoRawData (*C function*), 915
drivers_RT_RT700_MIMXRT798S.SDADC_GetDefaultConfig (*C function*), 908
drivers_RT_RT700_MIMXRT798S.SDADC_GetFifoIntStatusFlags (*C function*), 914
drivers_RT_RT700_MIMXRT798S.SDADC_GetFifoWatermarkStatusFlags (*C function*), 914
drivers_RT_RT700_MIMXRT798S.SDADC_Init (*C function*), 908
drivers_RT_RT700_MIMXRT798S.sdadc_ldo_drive_capability_t (*C type*), 919
drivers_RT_RT700_MIMXRT798S.sdadc_ref_mode_t (*C type*), 920
drivers_RT_RT700_MIMXRT798S.SDADC_RSTS (*C macro*), 839
drivers_RT_RT700_MIMXRT798S.sdadc_sample_rate_t (*C type*), 920
drivers_RT_RT700_MIMXRT798S.sdadc_volume_control_t (*C type*), 920
drivers_RT_RT700_MIMXRT798S.sdadc_watermark_t (*C type*), 920
drivers_RT_RT700_MIMXRT798S.SDADCFCLKDIV_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.SDADCFCLKSEL_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.SDIO0FCLKDIV_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.SDIO0FCLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.SDIO1FCLKDIV_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.SDIO1FCLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.SDK_ALIGN (*C macro*), 541
drivers_RT_RT700_MIMXRT798S.SDK_ATOMIC_LOCAL_ADD (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.SDK_ATOMIC_LOCAL_CLEAR (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.SDK_ATOMIC_LOCAL_CLEAR_AND_SET (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.SDK_ATOMIC_LOCAL_COMPARE_AND_SET (*C macro*), 544
drivers_RT_RT700_MIMXRT798S.SDK_ATOMIC_LOCAL_SET (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.SDK_ATOMIC_LOCAL_SUB (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.SDK_ATOMIC_LOCAL_TEST_AND_SET (*C macro*), 544
drivers_RT_RT700_MIMXRT798S.SDK_ATOMIC_LOCAL_TOGGLE (*C macro*), 543
drivers_RT_RT700_MIMXRT798S.SDK_DelayAtLeastUs (*C function*), 554
drivers_RT_RT700_MIMXRT798S.SDK_Free (*C function*), 553
drivers_RT_RT700_MIMXRT798S.SDK_IS_ALIGNED (*C macro*), 541, 542
drivers_RT_RT700_MIMXRT798S.SDK_ISR_EXIT_BARRIER (*C macro*), 541
drivers_RT_RT700_MIMXRT798S.SDK_L1DCACHE_ALIGN (*C macro*), 541
drivers_RT_RT700_MIMXRT798S.SDK_Malloc (*C function*), 553
drivers_RT_RT700_MIMXRT798S.SDK_SIZEALIGN (*C macro*), 541, 542
drivers_RT_RT700_MIMXRT798S.SDK_SIZEALIGN_DOWN (*C macro*), 541, 542
drivers_RT_RT700_MIMXRT798S.SDK_SIZEALIGN_UP (*C macro*), 541, 542
drivers_RT_RT700_MIMXRT798S.SEC_LOCK_PATTERN (*C macro*), 826
drivers_RT_RT700_MIMXRT798S.section_header (*C struct*), 882
drivers_RT_RT700_MIMXRT798S.section_type_t (*C type*), 881
drivers_RT_RT700_MIMXRT798S.secure_counter_t (*C type*), 210
drivers_RT_RT700_MIMXRT798S.SEMA42_BUSY_POLL_COUNT (*C macro*), 927
drivers_RT_RT700_MIMXRT798S.SEMA42_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.SEMA42_Deinit (*C function*), 925
drivers_RT_RT700_MIMXRT798S.SEMA42_GATE_NUM_RESET_ALL (*C macro*), 927
drivers_RT_RT700_MIMXRT798S.sema42_gate_status_t (*C type*), 925
drivers_RT_RT700_MIMXRT798S.SEMA42_GATEn (*C macro*), 927
drivers_RT_RT700_MIMXRT798S.SEMA42_GetGateStatus (*C function*), 926

drivers_RT_RT700_MIMXRT798S.SEMA42_Init (*C function*), 925
drivers_RT_RT700_MIMXRT798S.SEMA42_Lock (*C function*), 926
drivers_RT_RT700_MIMXRT798S.SEMA42_ResetAllGates (*C function*), 927
drivers_RT_RT700_MIMXRT798S.SEMA42_ResetGate (*C function*), 927
drivers_RT_RT700_MIMXRT798S.SEMA42_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.SEMA42_TryLock (*C function*), 925
drivers_RT_RT700_MIMXRT798S.SEMA42_Unlock (*C function*), 926
drivers_RT_RT700_MIMXRT798S.SENSE_CLKOUTCLKDIV_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.SENSE_CLKOUTCLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.SENSE_MAINCLKSEL_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.SENSEBASECLKSEL_OFFSET (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.SENSEDSPCPUCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.SENSEDSPCPUCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.SENSEMAINCLKDIV_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.SENSERAMCLKDIV_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.SENSERAMCLKSEL_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.SHA_MED_CCTRL0_OFFSET (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.SHA_MED_CSTAT0_OFFSET (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.SHA_MED_TCTRL0_OFFSET (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.SHA_MEDSEN_TSTAT0_OFFSET (*C macro*), 817
drivers_RT_RT700_MIMXRT798S.SHA_SEN_TCTRL0_OFFSET (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.size (*C var*), 282, 329
drivers_RT_RT700_MIMXRT798S.SLEEPCON_RCFG (*C macro*), 816
drivers_RT_RT700_MIMXRT798S.slots (*C var*), 538
drivers_RT_RT700_MIMXRT798S.SPI_RETRY_TIMES (*C macro*), 412, 661
drivers_RT_RT700_MIMXRT798S.srcAddr (*C var*), 282, 329
drivers_RT_RT700_MIMXRT798S.state (*C var*), 1051
drivers_RT_RT700_MIMXRT798S.status_t (*C type*), 553
drivers_RT_RT700_MIMXRT798S.SUPPRESS_FALL_THROUGH_WARNING (*C macro*), 556
drivers_RT_RT700_MIMXRT798S.SYSCON3_SEC_CLK_CTRL (*C macro*), 261
drivers_RT_RT700_MIMXRT798S.SYSPM_COUNT_STABLE_TIMEOUT_RETURN_VALUE (*C macro*), 931
drivers_RT_RT700_MIMXRT798S.SYSPM_Deinit (*C function*), 929
drivers_RT_RT700_MIMXRT798S.SYSPM_DisableCounter (*C function*), 930
drivers_RT_RT700_MIMXRT798S.syspm_event_t (*C type*), 929
drivers_RT_RT700_MIMXRT798S.SYSPM_GetEventCounter (*C function*), 931
drivers_RT_RT700_MIMXRT798S.SYSPM_GetInstructionCounter (*C function*), 931
drivers_RT_RT700_MIMXRT798S.SYSPM_Init (*C function*), 929
drivers_RT_RT700_MIMXRT798S.syspm_mode_t (*C type*), 929
drivers_RT_RT700_MIMXRT798S.syspm_monitor_t (*C type*), 929
drivers_RT_RT700_MIMXRT798S.SYSPM_ResetEvent (*C function*), 930
drivers_RT_RT700_MIMXRT798S.SYSPM_ResetInstructionEvent (*C function*), 930
drivers_RT_RT700_MIMXRT798S.SYSPM_SelectEvent (*C function*), 929
drivers_RT_RT700_MIMXRT798S.SYSPM_SetCountMode (*C function*), 930
drivers_RT_RT700_MIMXRT798S.SYSPM_SetStartStopControl (*C function*), 930
drivers_RT_RT700_MIMXRT798S.syspm_startstop_control_t (*C type*), 929
drivers_RT_RT700_MIMXRT798S.SYSTICKFCLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.SYSTICKFCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.tag (*C var*), 842
drivers_RT_RT700_MIMXRT798S.targetFreq (*C var*), 266
drivers_RT_RT700_MIMXRT798S.transferSize (*C var*), 1050
drivers_RT_RT700_MIMXRT798S.trim1DelayUs (*C var*), 266
drivers_RT_RT700_MIMXRT798S.trim2DelayUs (*C var*), 266
drivers_RT_RT700_MIMXRT798S.trng_clock_mode_t (*C type*), 934
drivers_RT_RT700_MIMXRT798S.trng_config_t (*C type*), 934
drivers_RT_RT700_MIMXRT798S.TRNG_Deinit (*C function*), 935
drivers_RT_RT700_MIMXRT798S.TRNG_GetDefaultConfig (*C function*), 934
drivers_RT_RT700_MIMXRT798S.TRNG_GetRandomData (*C function*), 935

drivers_RT_RT700_MIMXRT798S.TRNG_Init (*C function*), 935
drivers_RT_RT700_MIMXRT798S.trng_oscillator_mode_t (*C enum*), 934
drivers_RT_RT700_MIMXRT798S.trng_oscillator_mode_t (*C type*), 934
drivers_RT_RT700_MIMXRT798S.trng_oscillator_mode_t.kTRNG_DualOscillatorMode (*C enumerator*), 934
drivers_RT_RT700_MIMXRT798S.trng_oscillator_mode_t.kTRNG_SingleOscillatorModeOsc1 (*C enumerator*), 934
drivers_RT_RT700_MIMXRT798S.trng_oscillator_mode_t.kTRNG_SingleOscillatorModeOsc2 (*C enumerator*), 934
drivers_RT_RT700_MIMXRT798S.trng_ring_osc_div_t (*C type*), 934
drivers_RT_RT700_MIMXRT798S.trng_sample_mode_t (*C type*), 934
drivers_RT_RT700_MIMXRT798S.trng_statistical_check_limit_t (*C type*), 934
drivers_RT_RT700_MIMXRT798S.txDmaHandle (*C var*), 1050
drivers_RT_RT700_MIMXRT798S.U (*C var*), 840, 842
drivers_RT_RT700_MIMXRT798S.UART_RETRY_TIMES (*C macro*), 424, 691
drivers_RT_RT700_MIMXRT798S.UINT16_MAX (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.UINT32_MAX (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.UINT64_H (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.UINT64_L (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.UINTPTR_SIZE (*C macro*), 539
drivers_RT_RT700_MIMXRT798S.USB_CLOCKS (*C macro*), 258
drivers_RT_RT700_MIMXRT798S.USBCLKSRC24MCLKSEL_OFFSET (*C macro*), 263
drivers_RT_RT700_MIMXRT798S.USBFCLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.USBPHY_CLOCKS (*C macro*), 258
drivers_RT_RT700_MIMXRT798S.USDHC_AdjustDelayForManualTuning (*C function*), 945
drivers_RT_RT700_MIMXRT798S.USDHC_ADMA1_ADDRESS_ALIGN (*C macro*), 964
drivers_RT_RT700_MIMXRT798S.USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK (*C macro*), 965
drivers_RT_RT700_MIMXRT798S.USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT (*C macro*), 965
drivers_RT_RT700_MIMXRT798S.USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK (*C macro*), 965
drivers_RT_RT700_MIMXRT798S.USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT (*C macro*), 965
drivers_RT_RT700_MIMXRT798S.USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (*C macro*), 965
drivers_RT_RT700_MIMXRT798S.usdhc_adma1_descriptor_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.USDHC_ADMA1_LENGTH_ALIGN (*C macro*), 964
drivers_RT_RT700_MIMXRT798S.USDHC_ADMA2_ADDRESS_ALIGN (*C macro*), 964
drivers_RT_RT700_MIMXRT798S.USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK (*C macro*), 966
drivers_RT_RT700_MIMXRT798S.USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT (*C macro*), 965
drivers_RT_RT700_MIMXRT798S.USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (*C macro*), 966
drivers_RT_RT700_MIMXRT798S.usdhc_adma2_descriptor_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.USDHC_ADMA2_LENGTH_ALIGN (*C macro*), 965
drivers_RT_RT700_MIMXRT798S.usdhc_adma_config_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.USDHC_AssertHardwareReset (*C function*), 942
drivers_RT_RT700_MIMXRT798S.usdhc_boot_config_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.usdhc_boot_mode_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.usdhc_burst_len_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.usdhc_capability_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.usdhc_card_command_type_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.usdhc_card_response_type_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.USDHC_CardDetectByData3 (*C function*), 942
drivers_RT_RT700_MIMXRT798S.USDHC_CheckStdTuningResult (*C function*), 946
drivers_RT_RT700_MIMXRT798S.USDHC_CheckTuningError (*C function*), 946

drivers_RT_RT700_MIMXRT798S.USDHC_ClearInterruptStatusFlags (*C function*), 940
drivers_RT_RT700_MIMXRT798S.USDHC_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.usdhc_command_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.usdhc_config_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.USDHC_ConfigStrobeDLL (*C function*), 946
drivers_RT_RT700_MIMXRT798S.usdhc_data_bus_width_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.usdhc_data_t (*C type*), 964
drivers_RT_RT700_MIMXRT798S.USDHC_Deinit (*C function*), 938
drivers_RT_RT700_MIMXRT798S.USDHC_DetectCardInsert (*C function*), 943
drivers_RT_RT700_MIMXRT798S.USDHC_DisableInterruptSignal (*C function*), 940
drivers_RT_RT700_MIMXRT798S.USDHC_DisableInterruptStatus (*C function*), 940
drivers_RT_RT700_MIMXRT798S.usdhc_dma_mode_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.USDHC_DriverIRQHandler (*C function*), 948
drivers_RT_RT700_MIMXRT798S.USDHC_EnableAutoTuning (*C function*), 944
drivers_RT_RT700_MIMXRT798S.USDHC_EnableAutoTuningForCmdAndData (*C function*), 944
drivers_RT_RT700_MIMXRT798S.USDHC_EnableDDRMode (*C function*), 946
drivers_RT_RT700_MIMXRT798S.USDHC_EnableHS400Mode (*C function*), 946
drivers_RT_RT700_MIMXRT798S.USDHC_EnableInternalDMA (*C function*), 939
drivers_RT_RT700_MIMXRT798S.USDHC_EnableInterruptSignal (*C function*), 940
drivers_RT_RT700_MIMXRT798S.USDHC_EnableInterruptStatus (*C function*), 939
drivers_RT_RT700_MIMXRT798S.USDHC_EnableManualTuning (*C function*), 944
drivers_RT_RT700_MIMXRT798S.USDHC_EnableMmcBoot (*C function*), 943
drivers_RT_RT700_MIMXRT798S.USDHC_EnableSdioControl (*C function*), 943
drivers_RT_RT700_MIMXRT798S.USDHC_EnableStandardTuning (*C function*), 945
drivers_RT_RT700_MIMXRT798S.USDHC_EnableStrobeDLL (*C function*), 946
drivers_RT_RT700_MIMXRT798S.USDHC_EnableWakeupEvent (*C function*), 942
drivers_RT_RT700_MIMXRT798S.usdhc_endian_mode_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.USDHC_ForceClockOn (*C function*), 941
drivers_RT_RT700_MIMXRT798S.USDHC_GetAdmaErrorStatusFlags (*C function*), 941
drivers_RT_RT700_MIMXRT798S.USDHC_GetAutoCommand12ErrorStatusFlags (*C function*), 940
drivers_RT_RT700_MIMXRT798S.USDHC_GetCapability (*C function*), 941
drivers_RT_RT700_MIMXRT798S.USDHC_GetEnabledInterruptStatusFlags (*C function*), 940
drivers_RT_RT700_MIMXRT798S.USDHC_GetExecuteStdTuningStatus (*C function*), 945
drivers_RT_RT700_MIMXRT798S.USDHC_GetInterruptStatusFlags (*C function*), 940
drivers_RT_RT700_MIMXRT798S.USDHC_GetPresentStatusFlags (*C function*), 941
drivers_RT_RT700_MIMXRT798S.USDHC_GetStrobeDLLStatus (*C function*), 947
drivers_RT_RT700_MIMXRT798S.USDHC_GetTuningDelayStatus (*C function*), 944
drivers_RT_RT700_MIMXRT798S.usdhc_handle_t (*C type*), 964
drivers_RT_RT700_MIMXRT798S.usdhc_host_t (*C type*), 964
drivers_RT_RT700_MIMXRT798S.USDHC_Init (*C function*), 937
drivers_RT_RT700_MIMXRT798S.USDHC_MAX_BLOCK_COUNT (*C macro*), 964
drivers_RT_RT700_MIMXRT798S.USDHC_ReadData (*C function*), 942
drivers_RT_RT700_MIMXRT798S.USDHC_RequestReTuning (*C function*), 944
drivers_RT_RT700_MIMXRT798S.USDHC_RequestStopAtBlockGap (*C function*), 943
drivers_RT_RT700_MIMXRT798S.USDHC_RequestTuningForSDR50 (*C function*), 944
drivers_RT_RT700_MIMXRT798S.USDHC_Reset (*C function*), 938
drivers_RT_RT700_MIMXRT798S.USDHC_ResetStrobeDLL (*C function*), 946
drivers_RT_RT700_MIMXRT798S.USDHC_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.usdhc_scatter_gather_data_list_t (*C type*), 963
drivers_RT_RT700_MIMXRT798S.usdhc_scatter_gather_data_t (*C type*), 964
drivers_RT_RT700_MIMXRT798S.usdhc_scatter_gather_transfer_t (*C type*), 964
drivers_RT_RT700_MIMXRT798S.USDHC_SendCommand (*C function*), 942
drivers_RT_RT700_MIMXRT798S.USDHC_SetADMA1Descriptor (*C function*), 939
drivers_RT_RT700_MIMXRT798S.USDHC_SetADMA2Descriptor (*C function*), 939
drivers_RT_RT700_MIMXRT798S.USDHC_SetAdmaTableConfig (*C function*), 938
drivers_RT_RT700_MIMXRT798S.USDHC_SetCardActive (*C function*), 941
drivers_RT_RT700_MIMXRT798S.USDHC_SetContinueRequest (*C function*), 943
drivers_RT_RT700_MIMXRT798S.USDHC_SetDataBusWidth (*C function*), 942

drivers_RT_RT700_MIMXRT798S.USDHC_SetDataConfig (*C function*), 947
drivers_RT_RT700_MIMXRT798S.USDHC_SetForceEvent (*C function*), 943
drivers_RT_RT700_MIMXRT798S.USDHC_SetInternalDmaConfig (*C function*), 938
drivers_RT_RT700_MIMXRT798S.USDHC_SetMmcBootConfig (*C function*), 943
drivers_RT_RT700_MIMXRT798S.USDHC_SetSdClock (*C function*), 941
drivers_RT_RT700_MIMXRT798S.USDHC_SetStandardTuningCounter (*C function*), 945
drivers_RT_RT700_MIMXRT798S.USDHC_SetStrobeDllOverride (*C function*), 946
drivers_RT_RT700_MIMXRT798S.USDHC_SetTuningDelay (*C function*), 944
drivers_RT_RT700_MIMXRT798S.usdhc_transfer_callback_t (*C type*), 964
drivers_RT_RT700_MIMXRT798S.usdhc_transfer_direction_t (*C type*), 962
drivers_RT_RT700_MIMXRT798S.usdhc_transfer_function_t (*C type*), 964
drivers_RT_RT700_MIMXRT798S.usdhc_transfer_t (*C type*), 964
drivers_RT_RT700_MIMXRT798S.USDHC_TransferBlocking (*C function*), 948
drivers_RT_RT700_MIMXRT798S.USDHC_TransferCreateHandle (*C function*), 947
drivers_RT_RT700_MIMXRT798S.USDHC_TransferHandleIRQ (*C function*), 948
drivers_RT_RT700_MIMXRT798S.USDHC_TransferNonBlocking (*C function*), 947
drivers_RT_RT700_MIMXRT798S.USDHC_WriteData (*C function*), 942
drivers_RT_RT700_MIMXRT798S.USEC_TO_COUNT (*C macro*), 540
drivers_RT_RT700_MIMXRT798S.user_app_boot_invoke_option_t (*C type*), 841
drivers_RT_RT700_MIMXRT798S.userCtx0 (*C var*), 821
drivers_RT_RT700_MIMXRT798S.userCtx1 (*C var*), 821
drivers_RT_RT700_MIMXRT798S.userData (*C var*), 1051
drivers_RT_RT700_MIMXRT798S.UTICK1CLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.UTICK1FCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.utick_callback_t (*C type*), 973
drivers_RT_RT700_MIMXRT798S.UTICK_ClearStatusFlags (*C function*), 972
drivers_RT_RT700_MIMXRT798S.UTICK_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.UTICK_Deinit (*C function*), 972
drivers_RT_RT700_MIMXRT798S.UTICK_GetStatusFlags (*C function*), 972
drivers_RT_RT700_MIMXRT798S.UTICK_HandleIRQ (*C function*), 973
drivers_RT_RT700_MIMXRT798S.UTICK_Init (*C function*), 972
drivers_RT_RT700_MIMXRT798S.utick_mode_t (*C type*), 973
drivers_RT_RT700_MIMXRT798S.UTICK_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.UTICK_SetTick (*C function*), 972
drivers_RT_RT700_MIMXRT798S.VDD12 (*C var*), 818
drivers_RT_RT700_MIMXRT798S.VDDN (*C var*), 818
drivers_RT_RT700_MIMXRT798S.VGPUCLKDIV_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.VGPUCLKSEL_OFFSET (*C macro*), 265
drivers_RT_RT700_MIMXRT798S.vsel (*C var*), 818
drivers_RT_RT700_MIMXRT798S.vsel0 (*C var*), 817
drivers_RT_RT700_MIMXRT798S.vsel1 (*C var*), 817
drivers_RT_RT700_MIMXRT798S.vsel2 (*C var*), 817
drivers_RT_RT700_MIMXRT798S.vsel3 (*C var*), 818
drivers_RT_RT700_MIMXRT798S.WAKE32KCLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.width (*C var*), 537, 790
drivers_RT_RT700_MIMXRT798S.wrapper (*C var*), 536
drivers_RT_RT700_MIMXRT798S.WWDT2FCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.WWDT3FCLKSEL_OFFSET (*C macro*), 262
drivers_RT_RT700_MIMXRT798S.WWDT_ClearStatusFlags (*C function*), 975
drivers_RT_RT700_MIMXRT798S.WWDT_CLOCKS (*C macro*), 259
drivers_RT_RT700_MIMXRT798S.wwdt_config_t (*C type*), 976
drivers_RT_RT700_MIMXRT798S.WWDT_Deinit (*C function*), 974
drivers_RT_RT700_MIMXRT798S.WWDT_Disable (*C function*), 974
drivers_RT_RT700_MIMXRT798S.WWDT_Enable (*C function*), 974
drivers_RT_RT700_MIMXRT798S.WWDT_FIRST_WORD_OF_REFRESH (*C macro*), 976
drivers_RT_RT700_MIMXRT798S.WWDT_GetDefaultConfig (*C function*), 973
drivers_RT_RT700_MIMXRT798S.WWDT_GetStatusFlags (*C function*), 974
drivers_RT_RT700_MIMXRT798S.WWDT_Init (*C function*), 974

drivers_RT_RT700_MIMXRT798S.WWDT_Refresh (*C function*), 976
drivers_RT_RT700_MIMXRT798S.WWDT_SECOND_WORD_OF_REFRESH (*C macro*), 976
drivers_RT_RT700_MIMXRT798S.WWDT_SetTimeoutValue (*C function*), 975
drivers_RT_RT700_MIMXRT798S.WWDT_SetWarningValue (*C function*), 975
drivers_RT_RT700_MIMXRT798S.WWDT_SetWindowValue (*C function*), 975
drivers_RT_RT700_MIMXRT798S.XCACHE_CleanCache (*C function*), 978
drivers_RT_RT700_MIMXRT798S.XCACHE_CleanCacheByRange (*C function*), 978
drivers_RT_RT700_MIMXRT798S.XCACHE_CleanInvalidateCache (*C function*), 978
drivers_RT_RT700_MIMXRT798S.XCACHE_CleanInvalidateCacheByRange (*C function*), 978
drivers_RT_RT700_MIMXRT798S.XCACHE_DisableCache (*C function*), 977
drivers_RT_RT700_MIMXRT798S.XCACHE_EnableCache (*C function*), 977
drivers_RT_RT700_MIMXRT798S.XCACHE_GetInstanceByAddr (*C function*), 977
drivers_RT_RT700_MIMXRT798S.XCACHE_InvalidateCache (*C function*), 977
drivers_RT_RT700_MIMXRT798S.XCACHE_InvalidateCacheByRange (*C function*), 977
drivers_RT_RT700_MIMXRT798S.XCACHE_LINESIZE_BYTE (*C macro*), 980
drivers_RT_RT700_MIMXRT798S.XSPI2FCLKDIV_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.XSPI2FCLKSEL_OFFSET (*C macro*), 264
drivers_RT_RT700_MIMXRT798S.xspi_ahb_access_config_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_ahb_alignment_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_ahb_read_error_info_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_ahb_request_suspend_info_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_ahb_request_suspend_state_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_ahb_split_size_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_ahb_sub_buffer_status_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_ahb_write_config_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_ahbBuffer_config_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.xspi_ahbBuffer_perf_monitor_result_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_ahbBuffer_sub_buffer_config_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.XSPI_BlockAccessAfterAhbWrite (*C function*), 1026
drivers_RT_RT700_MIMXRT798S.xspi_byte_order_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_cache64_policy_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_cache64_region_config_t (*C type*), 1003
drivers_RT_RT700_MIMXRT798S.XSPI_Cache64_SetRegionConfig (*C function*), 1035
drivers_RT_RT700_MIMXRT798S.XSPI_CheckAhbBufferPerfMonitorHitOverflow (*C function*), 1022
drivers_RT_RT700_MIMXRT798S.XSPI_CheckAhbBufferPerfMonitorMissOverflow (*C function*), 1023
drivers_RT_RT700_MIMXRT798S.XSPI_CheckAhbBufferPerfMonitorTimeCounterOverflow (*C function*), 1022
drivers_RT_RT700_MIMXRT798S.XSPI_CheckAhbReadAccessAsserted (*C function*), 1010
drivers_RT_RT700_MIMXRT798S.XSPI_CheckAhbWriteAccessAsserted (*C function*), 1010
drivers_RT_RT700_MIMXRT798S.XSPI_CheckAndClearError (*C function*), 1032
drivers_RT_RT700_MIMXRT798S.XSPI_CheckAndClearErrorTG (*C function*), 1003
drivers_RT_RT700_MIMXRT798S.XSPI_CheckDataLearningFailure (*C function*), 1007
drivers_RT_RT700_MIMXRT798S.XSPI_CheckFSMValid (*C function*), 1034
drivers_RT_RT700_MIMXRT798S.XSPI_CheckFSMValidTG (*C function*), 1013
drivers_RT_RT700_MIMXRT798S.XSPI_CheckGlobalConfigLocked (*C function*), 1005
drivers_RT_RT700_MIMXRT798S.XSPI_CheckIPAccessAsserted (*C function*), 1014
drivers_RT_RT700_MIMXRT798S.XSPI_CheckIPAccessAssertedTG (*C function*), 1013
drivers_RT_RT700_MIMXRT798S.XSPI_CheckIPAccessGranted (*C function*), 1033
drivers_RT_RT700_MIMXRT798S.XSPI_CheckIpReadTriggered (*C function*), 1033
drivers_RT_RT700_MIMXRT798S.XSPI_CheckIpReadTriggeredTG (*C function*), 1013
drivers_RT_RT700_MIMXRT798S.XSPI_CheckIpRequestGranted (*C function*), 1030
drivers_RT_RT700_MIMXRT798S.XSPI_CheckIpRequestGrantedTG (*C function*), 1009
drivers_RT_RT700_MIMXRT798S.XSPI_CheckIpWriteTriggered (*C function*), 1033
drivers_RT_RT700_MIMXRT798S.XSPI_CheckIpWriteTriggeredTG (*C function*), 1013
drivers_RT_RT700_MIMXRT798S.XSPI_CheckModuleEnabled (*C function*), 1004
drivers_RT_RT700_MIMXRT798S.XSPI_CheckRxBufferWaterMarkExceed (*C function*), 1032
drivers_RT_RT700_MIMXRT798S.XSPI_CheckRxBufferWaterMarkExceedTG (*C function*), 1012

drivers_RT_RT700_MIMXRT798S.XSPI_CheckSFPFradEnabled (*C function*), 1015
drivers_RT_RT700_MIMXRT798S.XSPI_CheckTxBuffLockOpen (*C function*), 1034
drivers_RT_RT700_MIMXRT798S.XSPI_CheckTxBuffLockOpenTG (*C function*), 1011
drivers_RT_RT700_MIMXRT798S.XSPI_ClearAhbAccessSeqPointer (*C function*), 1026
drivers_RT_RT700_MIMXRT798S.XSPI_ClearAhbBuffer (*C function*), 1020
drivers_RT_RT700_MIMXRT798S.XSPI_ClearCmdExecutionArbitrationStatusFlags (*C function*), 1031
drivers_RT_RT700_MIMXRT798S.XSPI_ClearCmdExecutionArbitrationStatusFlagsTG (*C function*), 1010
drivers_RT_RT700_MIMXRT798S.XSPI_ClearErrorStatusFlags (*C function*), 1030
drivers_RT_RT700_MIMXRT798S.XSPI_ClearErrorStatusFlagsTG (*C function*), 1009
drivers_RT_RT700_MIMXRT798S.XSPI_ClearIPAccessSeqPointer (*C function*), 1016
drivers_RT_RT700_MIMXRT798S.XSPI_ClearRxBuffer (*C function*), 1031
drivers_RT_RT700_MIMXRT798S.XSPI_ClearRxBufferTG (*C function*), 1012
drivers_RT_RT700_MIMXRT798S.XSPI_ClearTgAddrWriteStatus (*C function*), 1015
drivers_RT_RT700_MIMXRT798S.XSPI_ClearTxBuffer (*C function*), 1031
drivers_RT_RT700_MIMXRT798S.XSPI_ClearTxBufferTG (*C function*), 1011
drivers_RT_RT700_MIMXRT798S.XSPI_CLOCKS (*C macro*), 258
drivers_RT_RT700_MIMXRT798S.xspi_command_type_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.xspi_config_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_data_learning_config_t (*C type*), 999
drivers_RT_RT700_MIMXRT798S.xspi_data_learning_pad_select_t (*C type*), 999
drivers_RT_RT700_MIMXRT798S.xspi_ddr_data_aligned_clk_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.XSPI_Deinit (*C function*), 1004
drivers_RT_RT700_MIMXRT798S.xspi_descriptor_lock_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.xspi_device_addr_mode_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.xspi_device_config_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.xspi_device_ddr_config_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.xspi_device_interface_settings_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.xspi_device_interface_type_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.xspi_device_status_reg_info_t (*C type*), 999
drivers_RT_RT700_MIMXRT798S.XSPI_DisableAhbBufferPerfMonitor (*C function*), 1022
drivers_RT_RT700_MIMXRT798S.XSPI_DisableInterrupts (*C function*), 1034
drivers_RT_RT700_MIMXRT798S.XSPI_DisableInterruptsTG (*C function*), 1008
drivers_RT_RT700_MIMXRT798S.xspi_dll_config_t (*C type*), 999
drivers_RT_RT700_MIMXRT798S.xspi_dll_mode_t (*C type*), 999
drivers_RT_RT700_MIMXRT798S.xspi_dll_para_t (*C type*), 999
drivers_RT_RT700_MIMXRT798S.xspi_edma_callback_t (*C type*), 1049
drivers_RT_RT700_MIMXRT798S.xspi_edma_handle_t (*C type*), 1049
drivers_RT_RT700_MIMXRT798S.xspi_edma_transfer_nsize_t (*C type*), 1049
drivers_RT_RT700_MIMXRT798S.XSPI_EnableAhbBufferPerfMonitor (*C function*), 1022
drivers_RT_RT700_MIMXRT798S.XSPI_EnableAhbBufferWriteFlush (*C function*), 1020
drivers_RT_RT700_MIMXRT798S.XSPI_EnableAhbReadPrefetch (*C function*), 1026
drivers_RT_RT700_MIMXRT798S.XSPI_EnableAhbWriteTerminate (*C function*), 1029
drivers_RT_RT700_MIMXRT798S.XSPI_EnableDozeMode (*C function*), 1006
drivers_RT_RT700_MIMXRT798S.XSPI_EnableInterrupts (*C function*), 1034
drivers_RT_RT700_MIMXRT798S.XSPI_EnableInterruptsTG (*C function*), 1008
drivers_RT_RT700_MIMXRT798S.XSPI_EnableInvertedSerialClockOutput (*C function*), 1006
drivers_RT_RT700_MIMXRT798S.XSPI_EnableModule (*C function*), 1004
drivers_RT_RT700_MIMXRT798S.XSPI_EnableRxDMA (*C function*), 1029
drivers_RT_RT700_MIMXRT798S.XSPI_EnableRxDMATG (*C function*), 1008
drivers_RT_RT700_MIMXRT798S.XSPI_EnableTxDMA (*C function*), 1029
drivers_RT_RT700_MIMXRT798S.XSPI_EnableTxDMATG (*C function*), 1008
drivers_RT_RT700_MIMXRT798S.XSPI_EnableVariableLatency (*C function*), 1006
drivers_RT_RT700_MIMXRT798S.xspi_exclusive_access_lock_mode_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.xspi_flags_t (*C type*), 999
drivers_RT_RT700_MIMXRT798S.xspi_frad_config_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.xspi_frad_tg_config_t (*C type*), 1001

drivers_RT_RT700_MIMXRT798S.xspi_frad_transaction_info_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.XSPI_GetAhbBufferPerfMonitorResult (*C function*), 1022
drivers_RT_RT700_MIMXRT798S.XSPI_GetAhbBufferPerfMonitorTimeCounter (*C function*), 1023
drivers_RT_RT700_MIMXRT798S.XSPI_GetAhbRequestSuspendInfo (*C function*), 1026
drivers_RT_RT700_MIMXRT798S.XSPI_GetAhbSubBufferStatus (*C function*), 1021
drivers_RT_RT700_MIMXRT798S.XSPI_GetBusIdleStatus (*C function*), 1010
drivers_RT_RT700_MIMXRT798S.XSPI_GetBusIdleStatusTG (*C function*), 1009
drivers_RT_RT700_MIMXRT798S.XSPI_GetCmdExecutionArbitrationStatusFlags (*C function*), 1031
drivers_RT_RT700_MIMXRT798S.XSPI_GetCmdExecutionArbitrationStatusFlagsTG (*C function*), 1010
drivers_RT_RT700_MIMXRT798S.XSPI_GetDataLearningEdgeMatchSignature (*C function*), 1007
drivers_RT_RT700_MIMXRT798S.XSPI_GetDefaultConfig (*C function*), 1004
drivers_RT_RT700_MIMXRT798S.XSPI_GetErrorStatusFlags (*C function*), 1030
drivers_RT_RT700_MIMXRT798S.XSPI_GetErrorStatusFlagsTG (*C function*), 1009
drivers_RT_RT700_MIMXRT798S.XSPI_GetFradLastTransactionsInfo (*C function*), 1014
drivers_RT_RT700_MIMXRT798S.XSPI_GetInstance (*C function*), 1003
drivers_RT_RT700_MIMXRT798S.XSPI_GetInterruptStatusFlags (*C function*), 1030
drivers_RT_RT700_MIMXRT798S.XSPI_GetMdadErrorReason (*C function*), 1015
drivers_RT_RT700_MIMXRT798S.XSPI_GetRxBufferAvailableBytesCount (*C function*), 1033
drivers_RT_RT700_MIMXRT798S.XSPI_GetRxBufferAvailableBytesCountTG (*C function*), 1012
drivers_RT_RT700_MIMXRT798S.XSPI_GetRxBufferRemovedBytesCount (*C function*), 1033
drivers_RT_RT700_MIMXRT798S.XSPI_GetRxBufferRemovedBytesCountTG (*C function*), 1013
drivers_RT_RT700_MIMXRT798S.XSPI_GetRxFifoAddress (*C function*), 1030
drivers_RT_RT700_MIMXRT798S.XSPI_GetRxFifoAddressTG (*C function*), 1009
drivers_RT_RT700_MIMXRT798S.XSPI_GetSFMStatusRegValue (*C function*), 1028
drivers_RT_RT700_MIMXRT798S.XSPI_GetTgAddrWriteStatus (*C function*), 1016
drivers_RT_RT700_MIMXRT798S.XSPI_GetTxFifoAddress (*C function*), 1030
drivers_RT_RT700_MIMXRT798S.XSPI_GetTxFifoAddressTG (*C function*), 1009
drivers_RT_RT700_MIMXRT798S.xspi_handle_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.xspi_hyper_bus_x16_mode_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.XSPI_Init (*C function*), 1003
drivers_RT_RT700_MIMXRT798S.xspi_ip_access_config_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.XSPI_IP_RX_BUFFER_SIZE (*C macro*), 1003
drivers_RT_RT700_MIMXRT798S.XSPI_IP_TX_BUFFER_SIZE (*C macro*), 1003
drivers_RT_RT700_MIMXRT798S.xspi_lut_instr_t (*C type*), 999
drivers_RT_RT700_MIMXRT798S.XSPI_LUT_SEQ (*C macro*), 1003
drivers_RT_RT700_MIMXRT798S.xspi_mdad_config_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.xspi_mdad_error_reason_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.xspi_mdad_mask_type_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.xspi_otfad_prefetch_boundary_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.xspi_pad_t (*C type*), 999
drivers_RT_RT700_MIMXRT798S.xspi_ppw_flag_clear_policy_t (*C type*), 1002
drivers_RT_RT700_MIMXRT798S.XSPI_ReadBlocking (*C function*), 1035
drivers_RT_RT700_MIMXRT798S.XSPI_ReadBlockingTG (*C function*), 1018
drivers_RT_RT700_MIMXRT798S.XSPI_ReadRxBuffer (*C function*), 1032
drivers_RT_RT700_MIMXRT798S.XSPI_ReadRxBufferTG (*C function*), 1012
drivers_RT_RT700_MIMXRT798S.XSPI_ResetSfmAndAhbDomain (*C function*), 1004
drivers_RT_RT700_MIMXRT798S.XSPI_ResetTgQueueTG (*C function*), 1005
drivers_RT_RT700_MIMXRT798S.XSPI_ResetTxRxBuffer (*C function*), 1031
drivers_RT_RT700_MIMXRT798S.XSPI_ResetTxRxBufferTG (*C function*), 1011
drivers_RT_RT700_MIMXRT798S.XSPI_ReturnAhbReadErrorInfo (*C function*), 1026
drivers_RT_RT700_MIMXRT798S.XSPI_RSTS (*C macro*), 840
drivers_RT_RT700_MIMXRT798S.xspi_sample_clk_config_t (*C type*), 999
drivers_RT_RT700_MIMXRT798S.xspi_sample_clk_source_t (*C type*), 999
drivers_RT_RT700_MIMXRT798S.xspi_secure_attribute_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.XSPI_SelectPPWFlagClearPolicy (*C function*), 1027
drivers_RT_RT700_MIMXRT798S.XSPI_SetAhbAccessBoundary (*C function*), 1023

drivers_RT_RT700_MIMXRT798S.XSPI_SetAhbAccessConfig (*C function*), 1029
drivers_RT_RT700_MIMXRT798S.XSPI_SetAhbAccessSplitSize (*C function*), 1023
drivers_RT_RT700_MIMXRT798S.XSPI_SetAhbBufferConfig (*C function*), 1020
drivers_RT_RT700_MIMXRT798S.XSPI_SetAhbErrorPayload (*C function*), 1025
drivers_RT_RT700_MIMXRT798S.XSPI_SetAhbReadDataSeqId (*C function*), 1024
drivers_RT_RT700_MIMXRT798S.XSPI_SetAhbReadStatusRegSeqId (*C function*), 1028
drivers_RT_RT700_MIMXRT798S.XSPI_SetAhbWriteDataSeqId (*C function*), 1024
drivers_RT_RT700_MIMXRT798S.XSPI_SetDataLearningConfig (*C function*), 1007
drivers_RT_RT700_MIMXRT798S.XSPI_SetDeviceConfig (*C function*), 1008
drivers_RT_RT700_MIMXRT798S.XSPI_SetHyperBusX16Mode (*C function*), 1005
drivers_RT_RT700_MIMXRT798S.XSPI_SetIpAccessConfig (*C function*), 1017
drivers_RT_RT700_MIMXRT798S.XSPI_SetSFMStatusRegInfo (*C function*), 1028
drivers_RT_RT700_MIMXRT798S.XSPI_SetSFPFradEALMode (*C function*), 1034
drivers_RT_RT700_MIMXRT798S.XSPI_SetSFPFradEALModeTG (*C function*), 1014
drivers_RT_RT700_MIMXRT798S.XSPI_SetSignalOutputValue (*C function*), 1006
drivers_RT_RT700_MIMXRT798S.xspi_sfp_frad_config_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.XSPI_SFP_FRAD_COUNT (*C macro*), 1003
drivers_RT_RT700_MIMXRT798S.xspi_sfp_mdad_config_t (*C type*), 1000
drivers_RT_RT700_MIMXRT798S.XSPI_SoftwareReset (*C function*), 1005
drivers_RT_RT700_MIMXRT798S.XSPI_StartAhbBufferPerfMonitor (*C function*), 1021
drivers_RT_RT700_MIMXRT798S.XSPI_StartIpAccess (*C function*), 1016
drivers_RT_RT700_MIMXRT798S.XSPI_StartIpAccessNonBlocking (*C function*), 1017
drivers_RT_RT700_MIMXRT798S.XSPI_StopAhbBufferPerfMonitor (*C function*), 1022
drivers_RT_RT700_MIMXRT798S.xspi_subbuffer_division_factor_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.XSPI_TARGET_GROUP_COUNT (*C macro*), 1003
drivers_RT_RT700_MIMXRT798S.xspi_target_group_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.xspi_tg_add_write_status_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.XSPI_TG_REG_ADDR (*C macro*), 1003
drivers_RT_RT700_MIMXRT798S.XSPI_TG_REG_ADDR_INT_EN (*C macro*), 1003
drivers_RT_RT700_MIMXRT798S.XSPI_TG_REG_ADDR_RBDR (*C macro*), 1003
drivers_RT_RT700_MIMXRT798S.XSPI_TG_REG_ADDR_RSER (*C macro*), 1003
drivers_RT_RT700_MIMXRT798S.XSPI_TG_REG_VAL (*C macro*), 1003
drivers_RT_RT700_MIMXRT798S.XSPI_TG_REG_VAL_RBDR_INDEX (*C macro*), 1003
drivers_RT_RT700_MIMXRT798S.xspi_transfer_callback_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.xspi_transfer_t (*C type*), 1001
drivers_RT_RT700_MIMXRT798S.XSPI_TransferAbort (*C function*), 1020
drivers_RT_RT700_MIMXRT798S.XSPI_TransferAbortEDMA (*C function*), 1050
drivers_RT_RT700_MIMXRT798S.XSPI_TransferBlocking (*C function*), 1018
drivers_RT_RT700_MIMXRT798S.XSPI_TransferCreateHandle (*C function*), 1018
drivers_RT_RT700_MIMXRT798S.XSPI_TransferCreateHandleEDMA (*C function*), 1049
drivers_RT_RT700_MIMXRT798S.XSPI_TransferEDMA (*C function*), 1050
drivers_RT_RT700_MIMXRT798S.XSPI_TransferGetCount (*C function*), 1019
drivers_RT_RT700_MIMXRT798S.XSPI_TransferGetTransferCountEDMA (*C function*), 1050
drivers_RT_RT700_MIMXRT798S.XSPI_TransferNonBlocking (*C function*), 1019
drivers_RT_RT700_MIMXRT798S.XSPI_TransferUpdateSizeEDMA (*C function*), 1049
drivers_RT_RT700_MIMXRT798S.XSPI_TriggerRxBufferPopEvent (*C function*), 1032
drivers_RT_RT700_MIMXRT798S.XSPI_TriggerRxBufferPopEventTG (*C function*), 1012
drivers_RT_RT700_MIMXRT798S.XSPI_UnlockIpAccessArbitration (*C function*), 1032
drivers_RT_RT700_MIMXRT798S.XSPI_UnlockIpAccessArbitrationTG (*C function*), 1016
drivers_RT_RT700_MIMXRT798S.XSPI_UpdateAhbBufferSize (*C function*), 1021
drivers_RT_RT700_MIMXRT798S.XSPI_UpdateAhbHreadyTimeOutValue (*C function*), 1025
drivers_RT_RT700_MIMXRT798S.XSPI_UpdateByteOrder (*C function*), 1004
drivers_RT_RT700_MIMXRT798S.XSPI_UpdateDeviceAddrMode (*C function*), 1005
drivers_RT_RT700_MIMXRT798S.XSPI_UpdateDllValue (*C function*), 1007
drivers_RT_RT700_MIMXRT798S.XSPI_UpdateIPAccessTimeoutCounter (*C function*), 1016
drivers_RT_RT700_MIMXRT798S.XSPI_UpdateLUT (*C function*), 1004
drivers_RT_RT700_MIMXRT798S.XSPI_UpdatePageWaitTimeCounter (*C function*), 1027
drivers_RT_RT700_MIMXRT798S.XSPI_UpdateRxBufferWaterMark (*C function*), 1031

drivers_RT_RT700_MIMXRT798S.XSPI_UpdateRxBufferWaterMarkTG (*C function*), 1014
drivers_RT_RT700_MIMXRT798S.XSPI_UpdateSFPArbitrationLockTimeoutCounter (*C function*), 1015
drivers_RT_RT700_MIMXRT798S.XSPI_UpdateSFPConfig (*C function*), 1015
drivers_RT_RT700_MIMXRT798S.XSPI_UpdateTxBufferWaterMark (*C function*), 1032
drivers_RT_RT700_MIMXRT798S.XSPI_UpdateTxBufferWaterMarkTG (*C function*), 1011
drivers_RT_RT700_MIMXRT798S.XSPI_WriteBlocking (*C function*), 1034
drivers_RT_RT700_MIMXRT798S.XSPI_WriteBlockingTG (*C function*), 1017
drivers_RT_RT700_MIMXRT798S.XSPI_WriteTxBuffer (*C function*), 1032
drivers_RT_RT700_MIMXRT798S.XSPI_WriteTxBufferTG (*C function*), 1011

H

HAL, **1126**

I

Internal Trusted Storage, **1126**
ITS, **1126**

M

MPC, **1126**
MPU, **1126**

N

NSPE, **1126**

P

PPC, **1126**
Protected Storage, **1126**
PS, **1126**
PSA, **1126**
PSA RoT, **1126**
PSA-FF, **1126**
PSA-FF-M, **1126**

R

RoT, **1126**
RoT Service, **1127**

S

S/NS, **1127**
SAU, **1127**
Secure Partition, **1127**
Secure Partition Manager, **1127**
SFN, **1127**
SP, **1127**
SPE, **1127**
SPM, **1127**
SPRT, **1127**
SPRTL, **1127**
SS, **1127**
SVC, **1127**

T

TBSA-M, **1127**
TF-M, **1127**
TFM, **1127**