MCUXpresso SDK Documentation

Release 26.09.00

NXP
Sep 29, 2026

Table of contents

1 MIMXRT685-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 . . . 40
  - 1.4.1 Getting Started with MCUXpresso SDK Repository . . . 40
- 1.5 Getting Started with MCUXpresso SDK Xplorer . . . 68
  - 1.5.1 Getting Started with MCUXpresso SDK Xplorer . . . 68
- 1.6 Release Notes . . . 110
  - 1.6.1 MCUXpresso SDK Release Notes . . . 111
- 1.7 ChangeLog . . . 117
  - 1.7.1 MCUXpresso SDK Changelog . . . 117
- 1.8 Driver API Reference Manual . . . 205
- 1.9 Middleware Documentation . . . 205
  - 1.9.1 Multicore . . . 205
  - 1.9.2 MCU Boot . . . 205
  - 1.9.3 eIQ . . . 205
  - 1.9.4 FreeMASTER . . . 205
  - 1.9.5 NXP Wi-Fi . . . 206
  - 1.9.6 FreeRTOS . . . 206
  - 1.9.7 lwIP . . . 206
  - 1.9.8 File systemFatfs . . . 206
  - 1.9.9 DSP Audio Streamer . . . 206

2 MIMXRT685S **207**
- 2.1 ACMP: Analog Comparator Driver . . . 207
- 2.2 CACHE: CACHE Memory Controller . . . 214
- 2.3 CASPER: The Cryptographic Accelerator and Signal Processing Engine with RAM sharing . . . 218
- 2.4 casper_driver . . . 218
- 2.5 casper_driver_pkha . . . 222
- 2.6 Clock Driver . . . 225
- 2.7 CRC: Cyclic Redundancy Check Driver . . . 263
- 2.8 CTIMER: Standard counter/timers . . . 266
- 2.9 DMA: Direct Memory Access Controller Driver . . . 277
- 2.10 DMIC: Digital Microphone . . . 295
- 2.11 DMIC DMA Driver . . . 295
- 2.12 DMIC Driver . . . 298
- 2.13 DSP Driver . . . 308
- 2.14 FLEXCOMM: FLEXCOMM Driver . . . 309
- 2.15 FLEXCOMM Driver . . . 309
- 2.16 FLEXSPI: Flexible Serial Peripheral Interface Driver . . . 310
- 2.17 FLEXSPI DMA Driver . . . 330
- 2.18 FMEAS: Frequency Measure Driver . . . 332

2.19 Hashcrypt: The Cryptographic Accelerator . . . 333
2.20 Hashcrypt Background HASH . . . 333
2.21 Hashcrypt common functions . . . 334
2.22 Hashcrypt AES . . . 337
2.23 Hashcrypt HASH . . . 341
2.24 I2C: Inter-Integrated Circuit Driver . . . 343
2.25 I2C DMA Driver . . . 343
2.26 I2C Driver . . . 345
2.27 I2C Master Driver . . . 350
2.28 I2C Slave Driver . . . 360
2.29 I2S: I2S Driver . . . 370
2.30 I2S_BRIDGE: I2S bridging and signal sharing configuration . . . 370
2.31 I2S DMA Driver . . . 373
2.32 I2S Driver . . . 376
2.33 I3C: I3C Driver . . . 386
2.34 I3C Common Driver . . . 389
2.35 I3C Master DMA Driver . . . 392
2.36 I3C Master Driver . . . 395
2.37 I3C Slave DMA Driver . . . 427
2.38 I3C Slave Driver . . . 429
2.39 IAP Boot Driver . . . 443
2.40 IAP: In Application Programming Driver . . . 444
2.41 IAP FlexSPI Driver . . . 444
2.42 IAP OTP Driver . . . 460
2.43 INPUTMUX: Input Multiplexing Driver . . . 462
2.44 IOPCTL: Input/Output Pad Controller . . . 485
2.45 Common Driver . . . 488
2.46 LPADC: 12-bit SAR Analog-to-Digital Converter Driver . . . 506
2.47 GPIO: General Purpose I/O . . . 527
2.48 MRT: Multi-Rate Timer . . . 531
2.49 MU: Messaging Unit . . . 536
2.50 OSTIMER: OS Event Timer Driver . . . 545
2.51 OTFAD: On The Fly AES-128 Decryption Driver . . . 550
2.52 PINT: Pin Interrupt and Pattern Match Driver . . . 553
2.53 Power Driver . . . 563
2.54 POWERQUAD: PowerQuad hardware accelerator . . . 579
2.55 PUF: Physical Unclonable Function . . . 612
2.56 Reset Driver . . . 615
2.57 RTC: Real Time Clock . . . 621
2.58 SCTimer: SCTimer/PWM (SCT) . . . 627
2.59 SEMA42: Hardware Semaphores Driver . . . 651
2.60 SPI: Serial Peripheral Interface Driver . . . 655
2.61 SPI DMA Driver . . . 655
2.62 SPI Driver . . . 659
2.63 TRNG: True Random Number Generator . . . 669
2.64 USART: Universal Synchronous/Asynchronous Receiver/Transmitter Driver . . . 675
2.65 USART DMA Driver . . . 675
2.66 USART Driver . . . 678
2.67 USDHC: Ultra Secured Digital Host Controller Driver . . . 695
2.68 UTICK: MictoTick Timer Driver . . . 730
2.69 WWDT: Windowed Watchdog Timer Driver . . . 731

3 Middleware **737**
3.1 Boot . . . 737
3.1.1 MCUboot in MCUXpresso SDK . . . 737
3.2 Connectivity . . . 738
3.2.1 lwIP . . . 738
3.3 File System . . . 752

3.3.1 FatFs . . . 752
3.4 Motor Control . . . 754
3.4.1 FreeMASTER . . . 754
3.5 Multicore SDK . . . 792
3.5.1 Quick links . . . 792
3.5.2 Components . . . 793
3.6 Multimedia . . . 882
3.6.1 Xtensa Audio Framework (XAF) . . . 882
3.7 Wireless . . . 897
3.7.1 NXP Wireless Framework and Stacks . . . 897

4 RTOS **959**
4.1 FreeRTOS . . . 959
4.1.1 FreeRTOS kernel . . . 959
4.1.2 FreeRTOS drivers . . . 963
4.1.3 backoffalgorithm . . . 963
4.1.4 corehttp . . . 963
4.1.5 corejson . . . 963
4.1.6 coremqtt . . . 963
4.1.7 corepkcs11 . . . 964
4.1.8 freertos-plus-tcp . . . 964

5 Downloads **965**
5.1 boards . . . 965
5.2 develop . . . 965

Index **967**

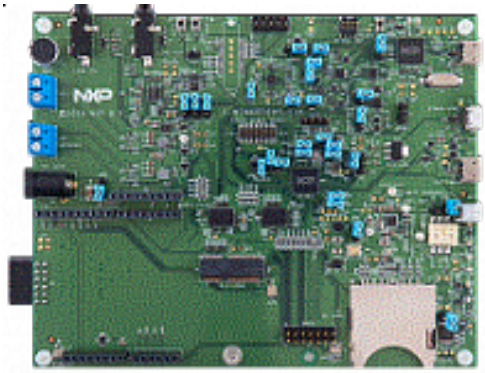
This documentation contains information specific to the evkmimxrt685 board.

Chapter 1

MIMXRT685-EVK

1.1 Overview

The NXP MIMXRT685-EVK is a development board for the i.MX RT600 Crossover MCU with Arm Cortex-M33 and DSP Cores.



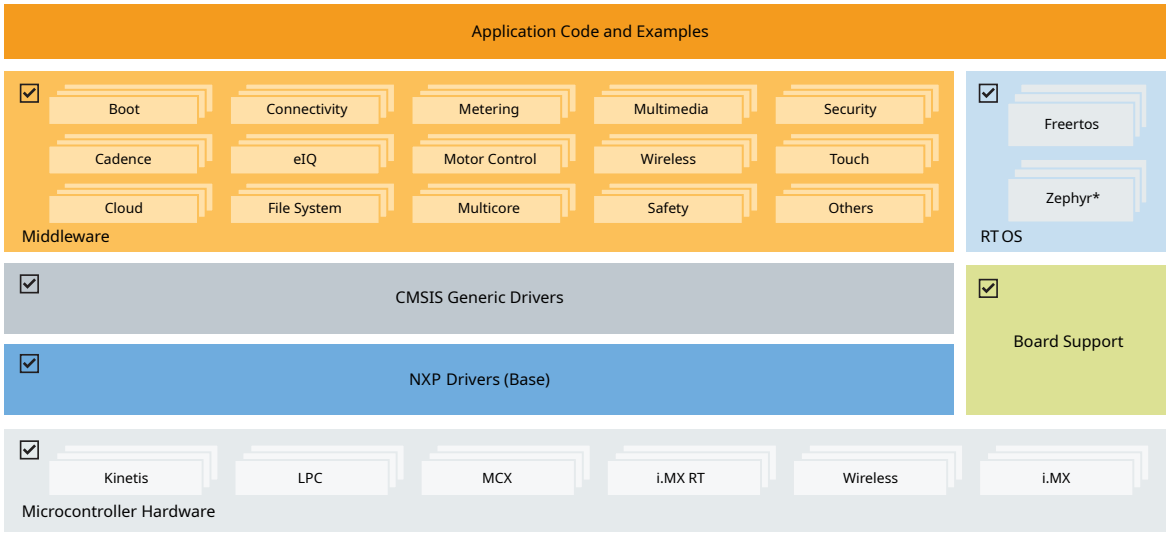
MCU device and part on board is shown below:

- Device: MIMXRT685S
- PartNumber: MIMXRT685SFVKB

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

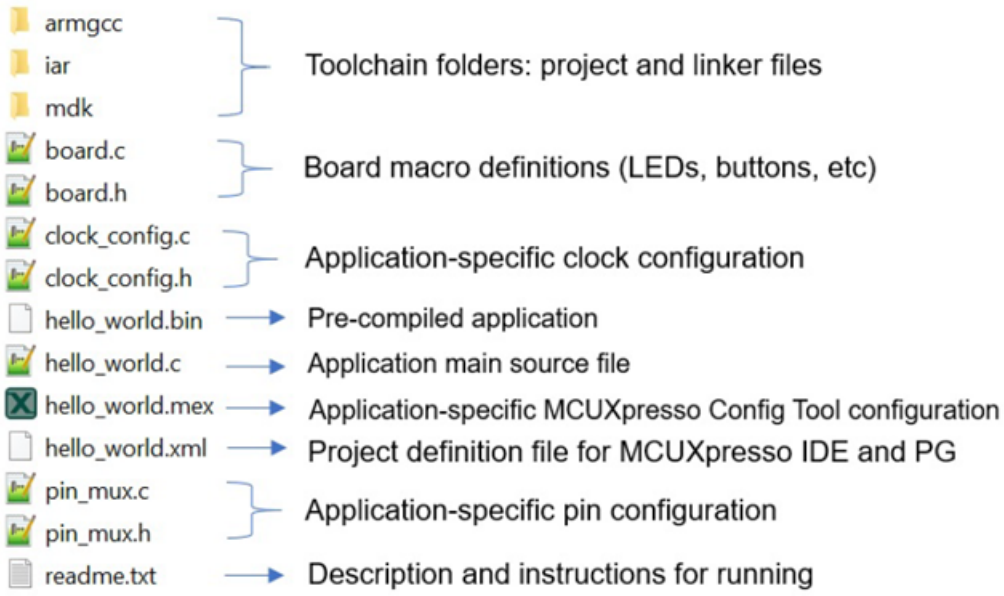
MCUXpresso SDK board support package provides example applications for NXP development and evaluation boards for Arm Cortex-M cores including Freedom, Tower System, i.mxrt EVK boards,and LPCXpresso boards. Board support packages are found inside the top-level boards folder and each supported board has 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):

- 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's 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).
- 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's RTOS drivers

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.

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>/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

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

Note: Ensure that the MCUXpresso IDE toolchain is included when generating the MCUXpresso SDK package.

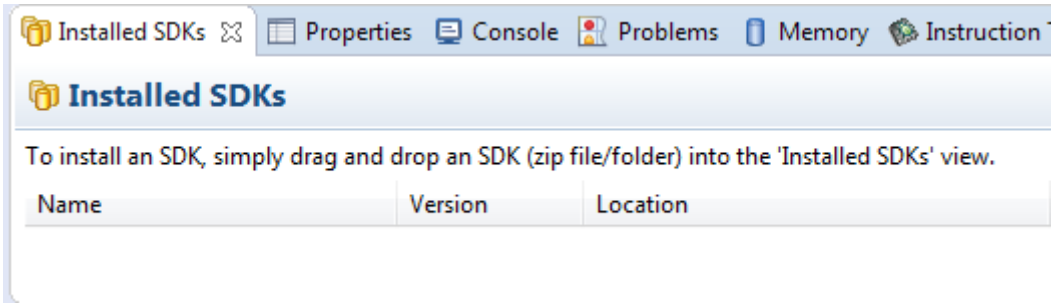
This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The hello_world demo application targeted for the MIMXRT685-EVK hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso 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 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 be located outside the MCUXpresso SDK tree.

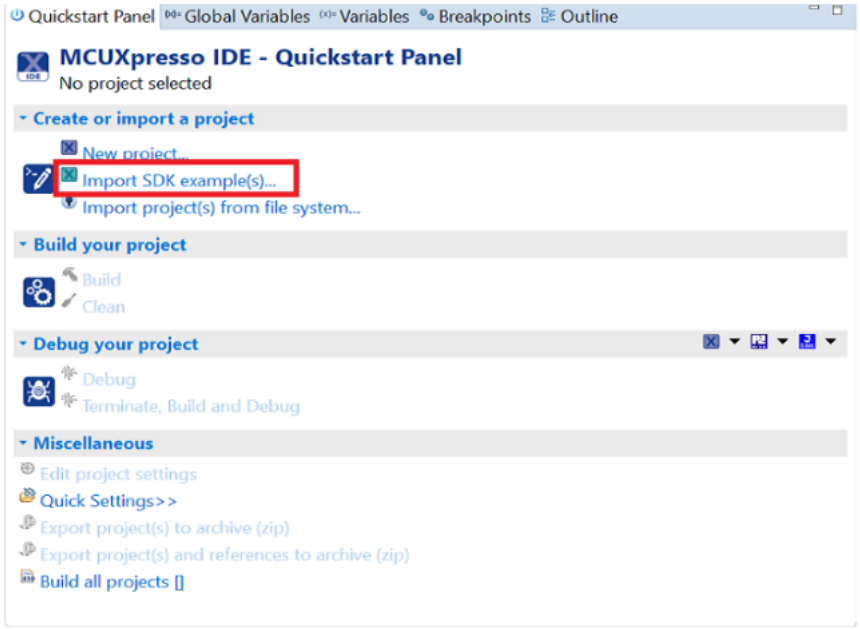
Parent topic:*Run a demo using MCUXpresso IDE*

Build an example application To build an example application, follow these steps.

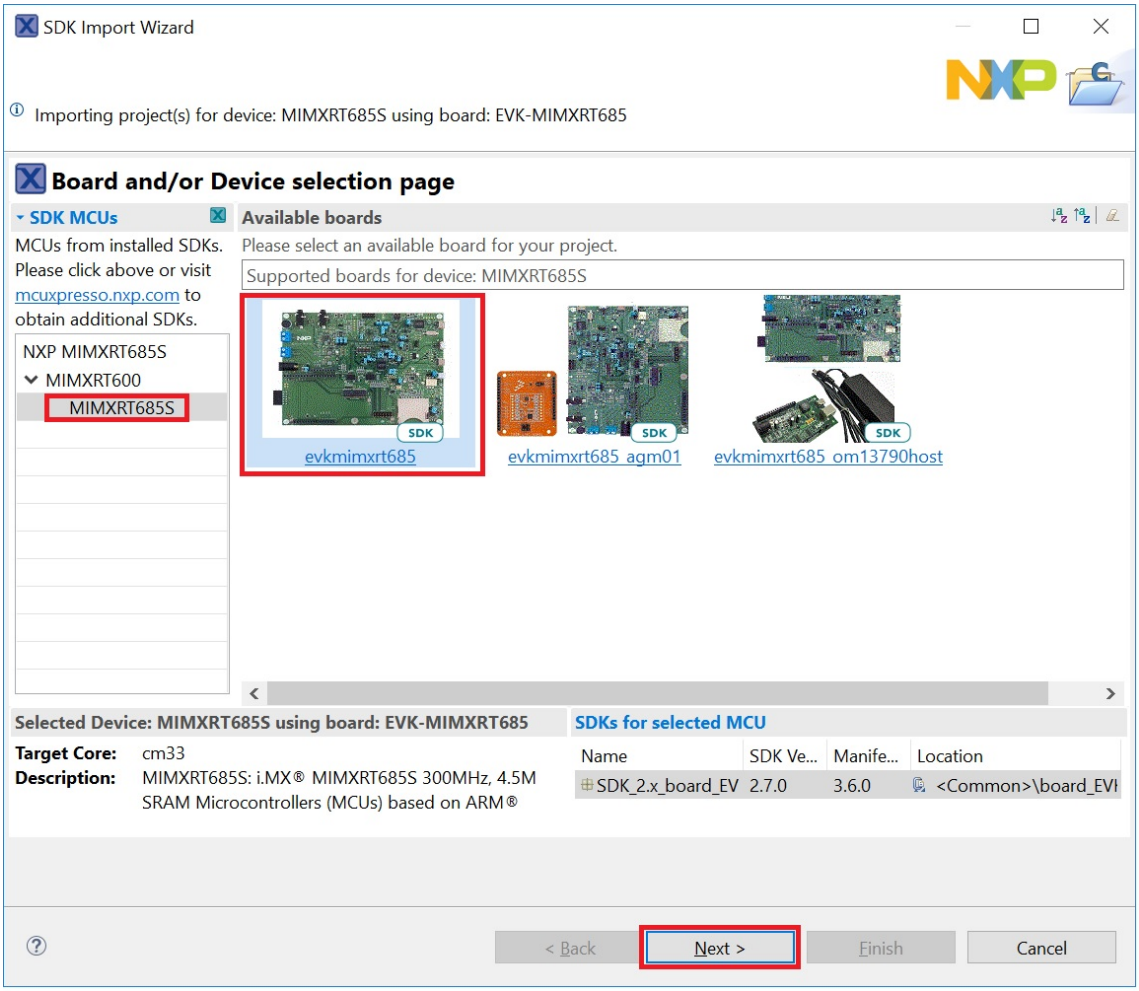
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



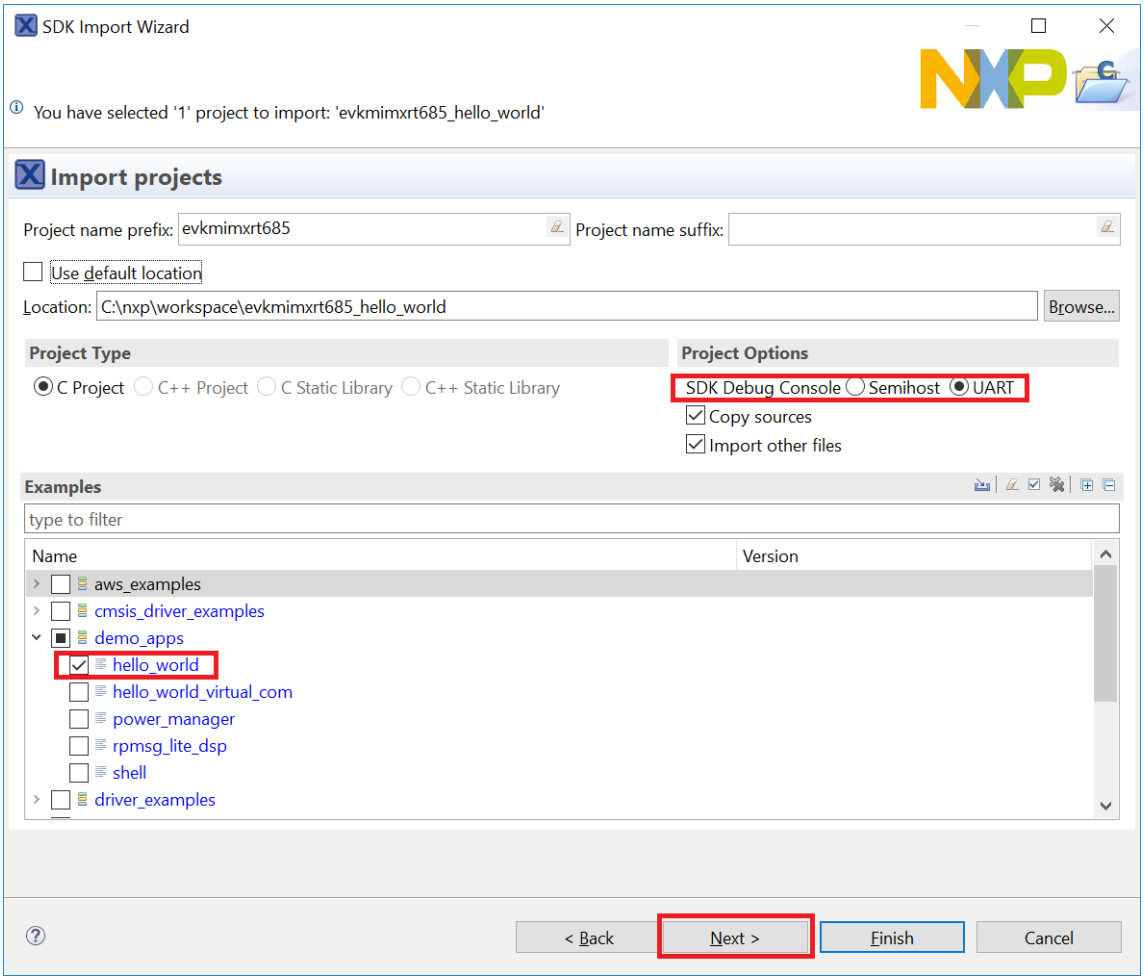
2. On the **Quickstart Panel**, click **Import SDK example(s)...**.



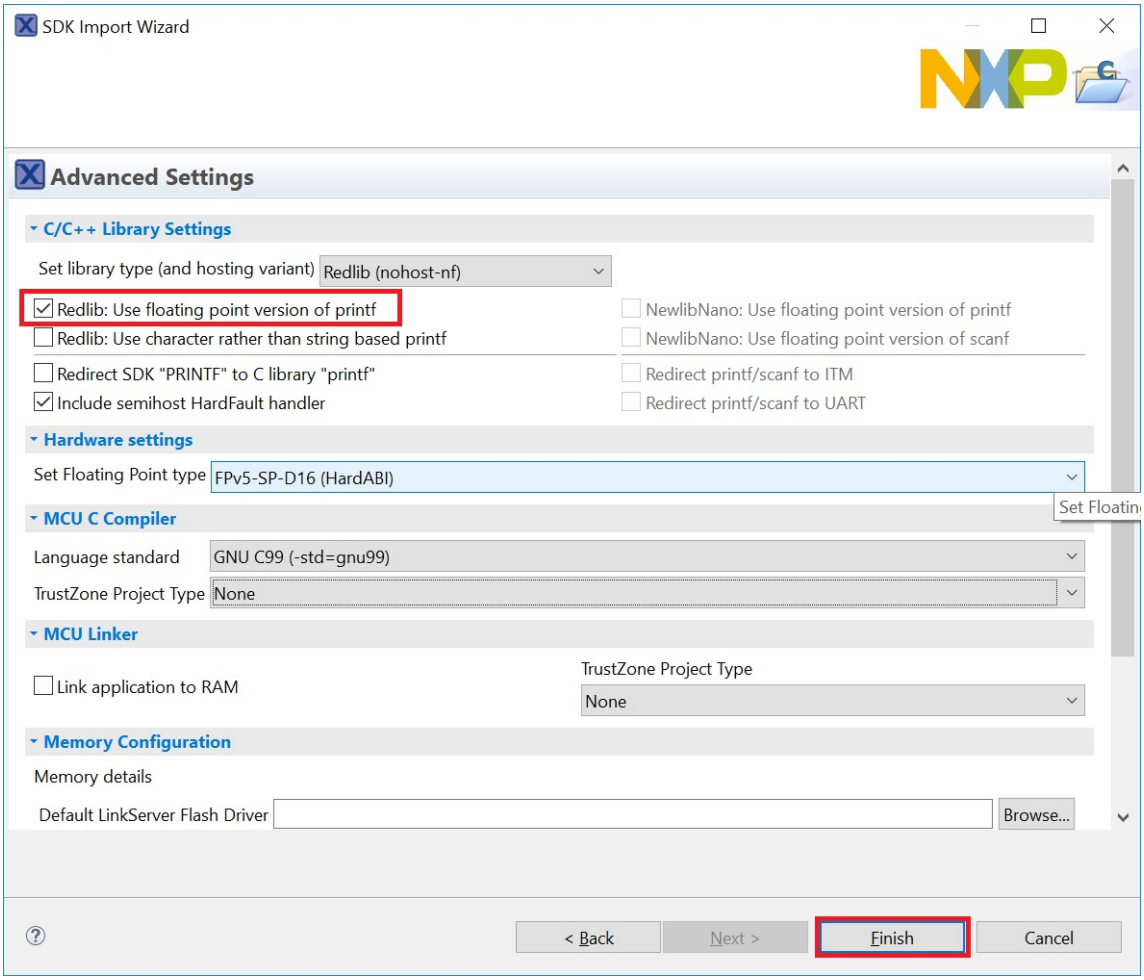
3. In the window that appears, expand the **MIMXRT600** folder and select **MIMXRT685S**. Then, select **evkmimxrt685** and click **Next**.



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



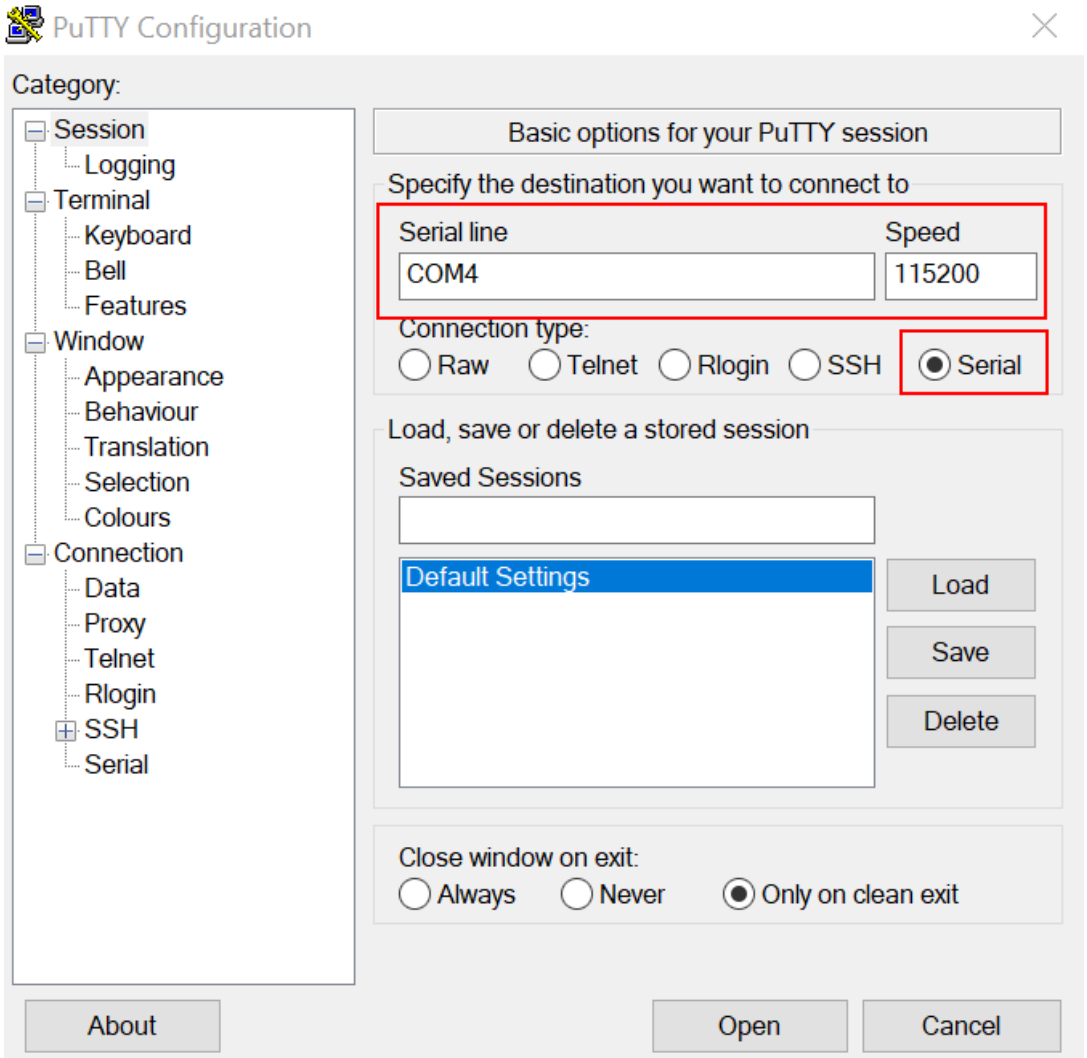
5. Ensure **Redlib: Use floating-point version of printf** is selected if the example prints floating-point numbers on the terminal. Otherwise, it is not necessary to select this option. Then, click **Finish**.



Run an example application For more information on debug probe support in the MCUXpresso IDE, see community.nxp.com.

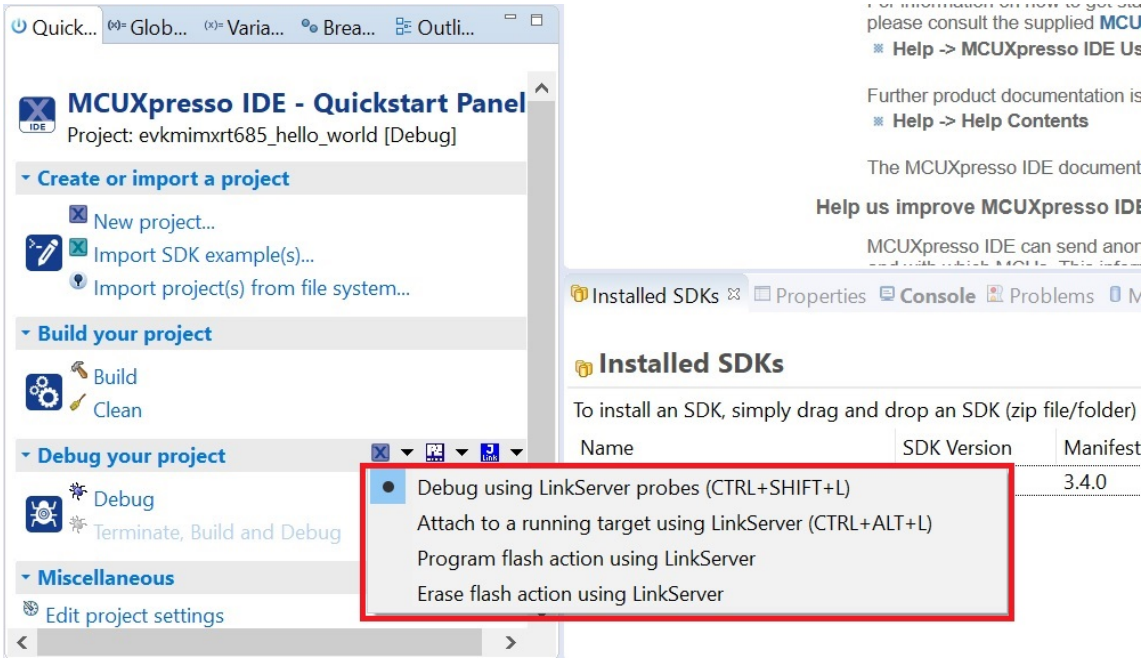
To download and run the application, perform the following steps:

1. See the table in Default debug interfaces to determine the debug interface that comes loaded on your specific hardware platform.
   - For boards with CMSIS-DAP/mbed/DAPLink interfaces, visit developer.mbed.org/handbook/Windows-serial-configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
   - For boards with a P&E Micro interface, see PE micro to download and install the P&E Micro Hardware Interface Drivers package.
2. Connect the development platform to your PC via a USB cable.
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:
   1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in board.h file)
   2. No parity
   3. 8 data bits

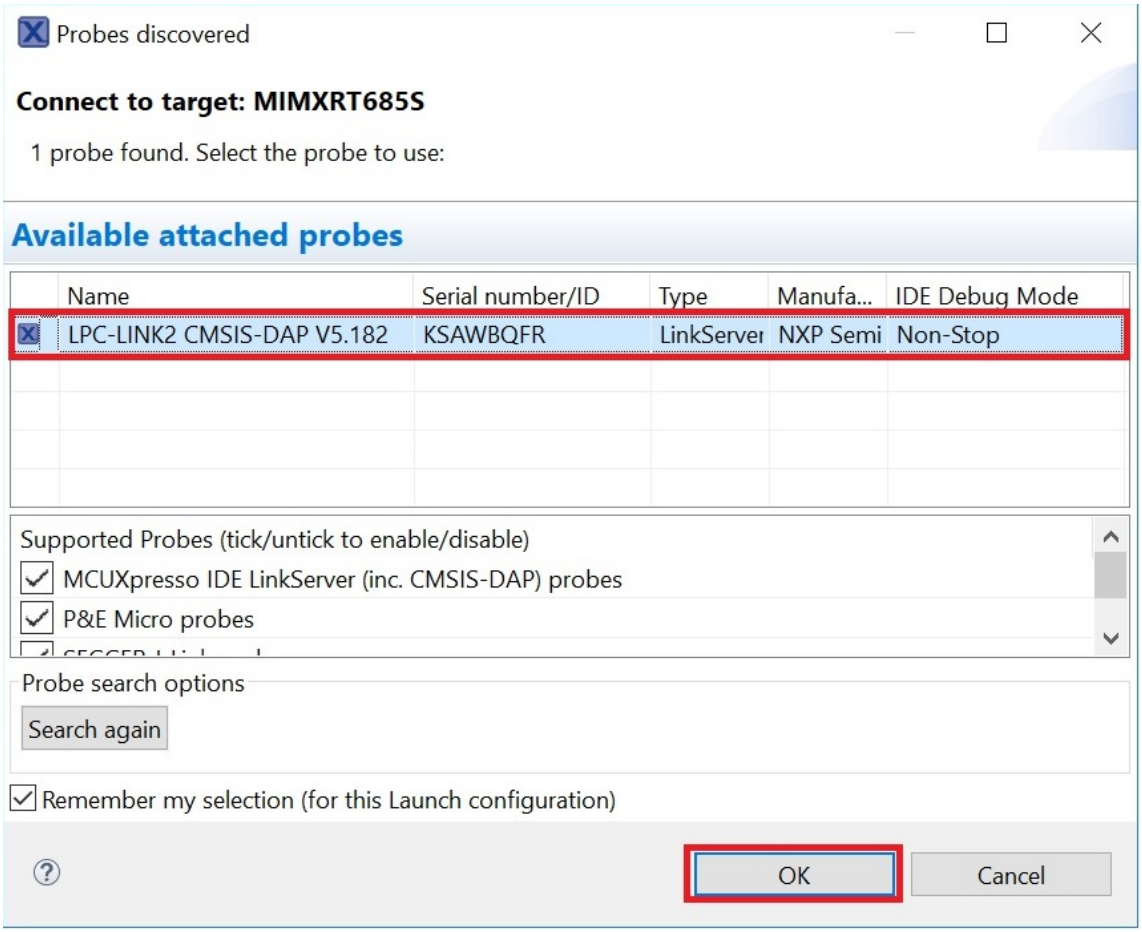


4. 1 stop bit

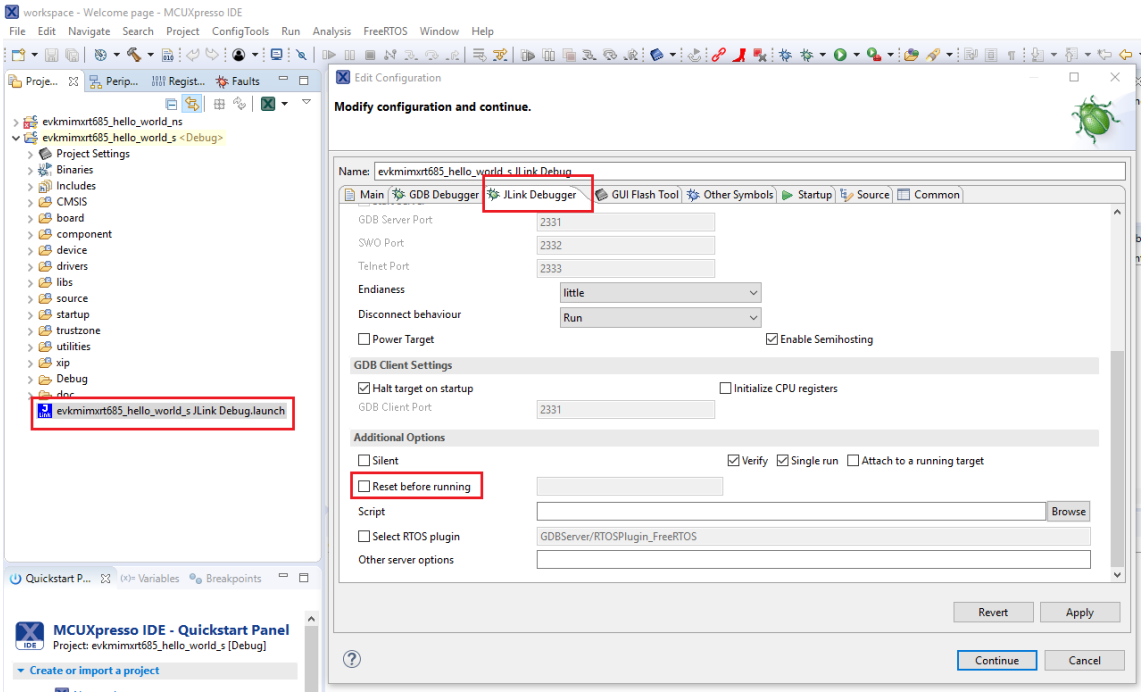
4. On the **Quickstart Panel**, click **Debug** evkmimxrt685_hello_world [Debug] to launch the debug session.



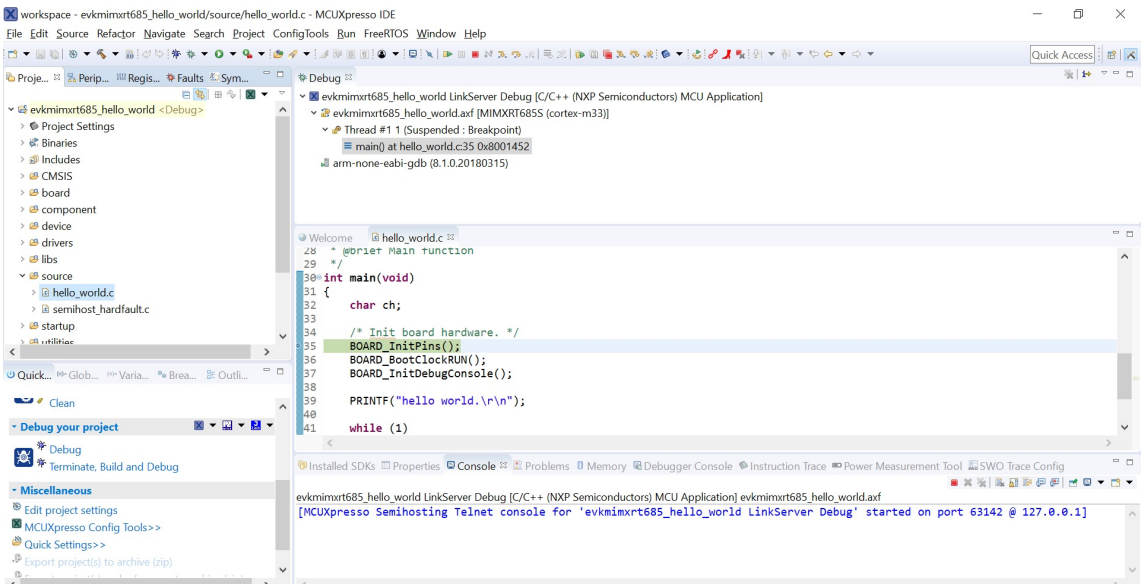
5. The first time you debug a project, the **Debug Emulator Selection** dialog is displayed, showing all supported probes that are attached to your computer. Select the probe through which you want to debug and click **OK**. (For any future debug sessions, the stored probe selection is automatically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



Note: If the debug probe is CMSIS-DAP and the debugging application is running in flash, make sure that the board is set to FlexSPI flash boot mode (ISP2: ISP1: ISP0 = ON, OFF, ON). Otherwise, you should not set to FlexSPI boot mode when debugging the application. If the debug probe is J-Link, **Reset before running** must be disabled. Double click the <example> J-Link Debug.launch file, and deselect this option under **JLink Debugger ->Additional Options** menu.

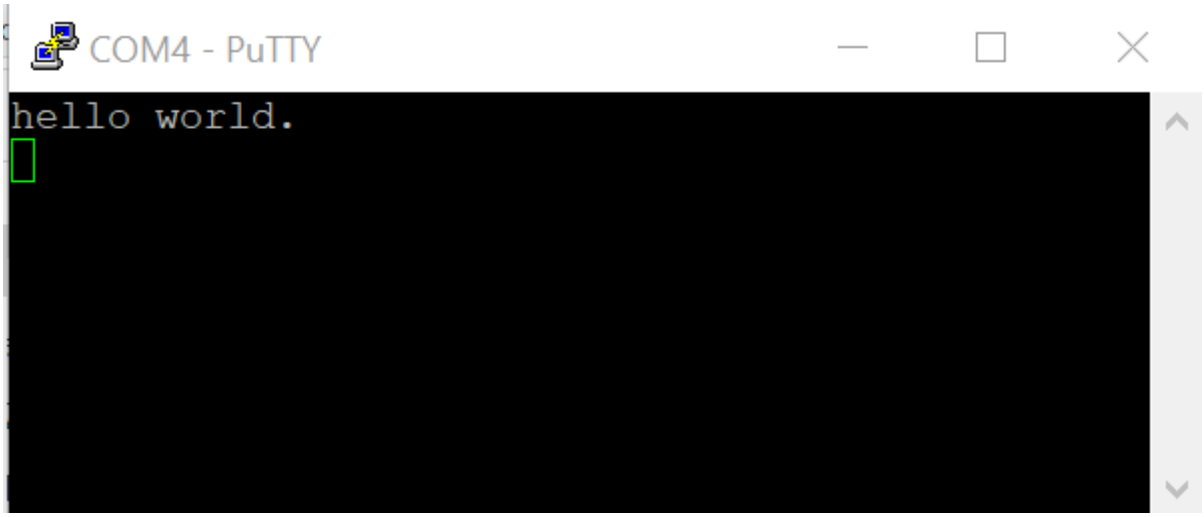


6. The application is downloaded to the target and automatically runs to main().



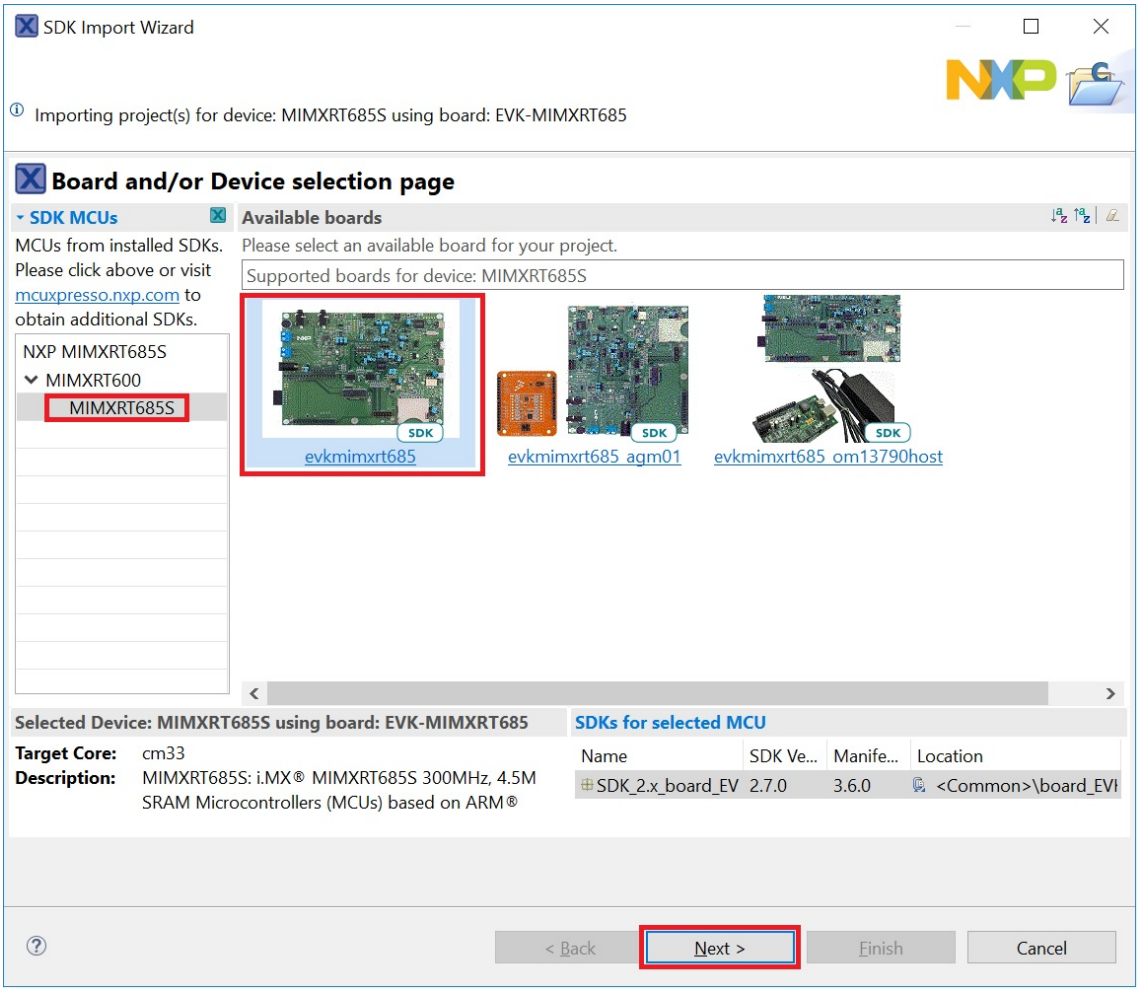
7. Start the application by clicking **Resume**.

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

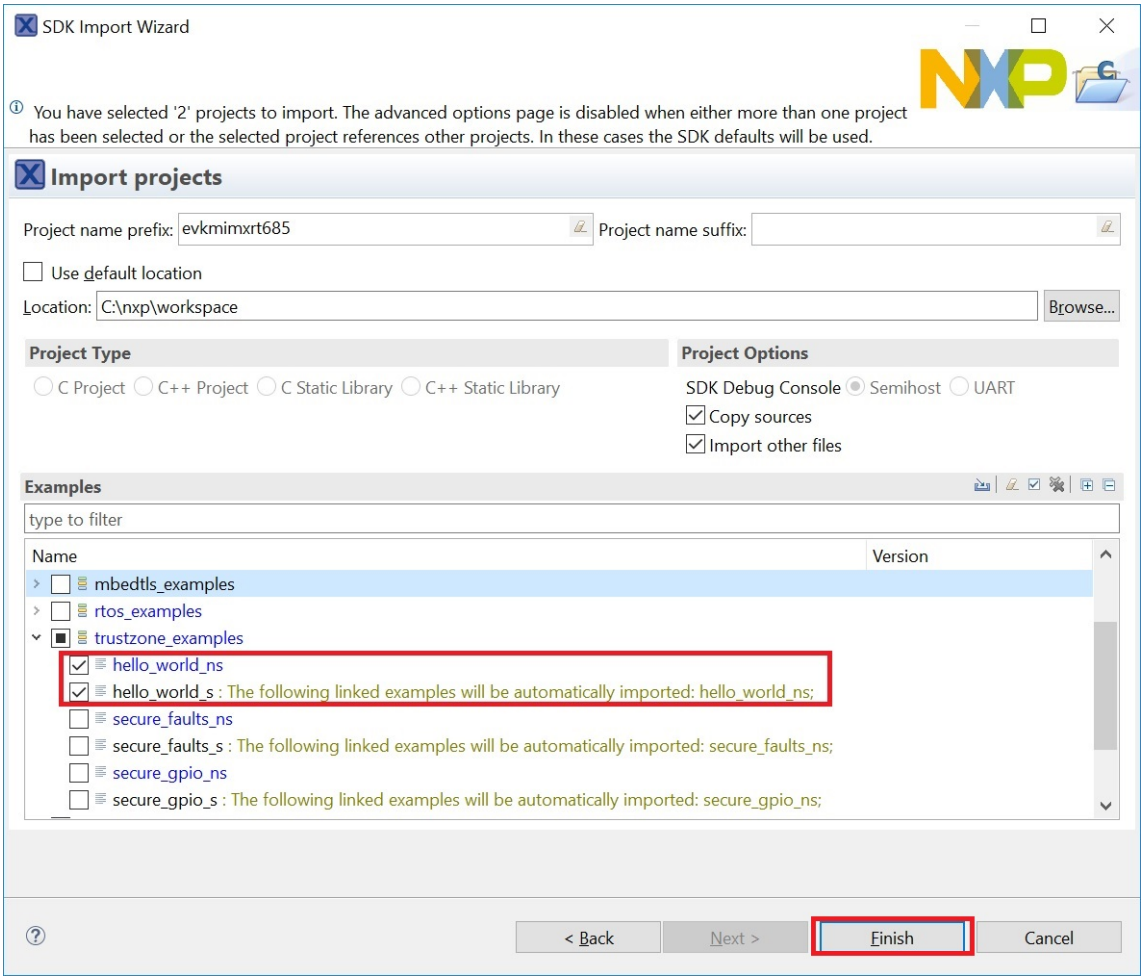


Build a TrustZone example application This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The TrustZone version of the hello_world example application targeted for the MIMXRT685-EVK hardware platform is used as an example, though these steps can be applied to any TrustZone example application in the MCUXpresso SDK.

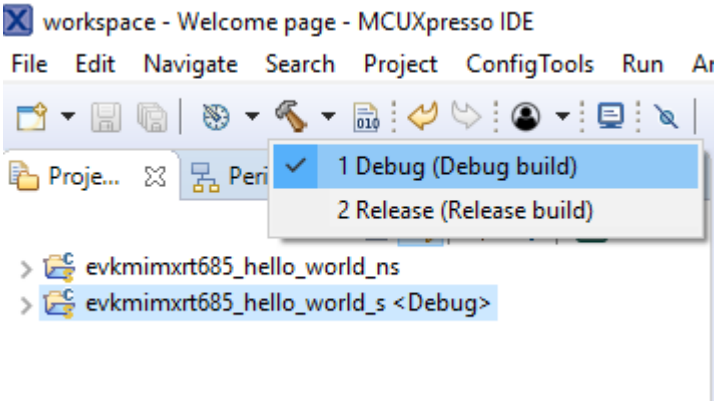
1. TrustZone examples are imported into the workspace in a similar way as single core applications. When the SDK zip package for MIMXRT685-EVK is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, expand the **MIMXRT600** folder and select **MIMXRT685S**. Then, select **evkmimxrt685** and click **Next**.



2. Expand the trustzone_examples/ folder and select hello_world_s. Because TrustZone examples are linked together, the non-secure project is automatically imported with the secure project, and there is no need to select it explicitly. Then, click **Finish**.



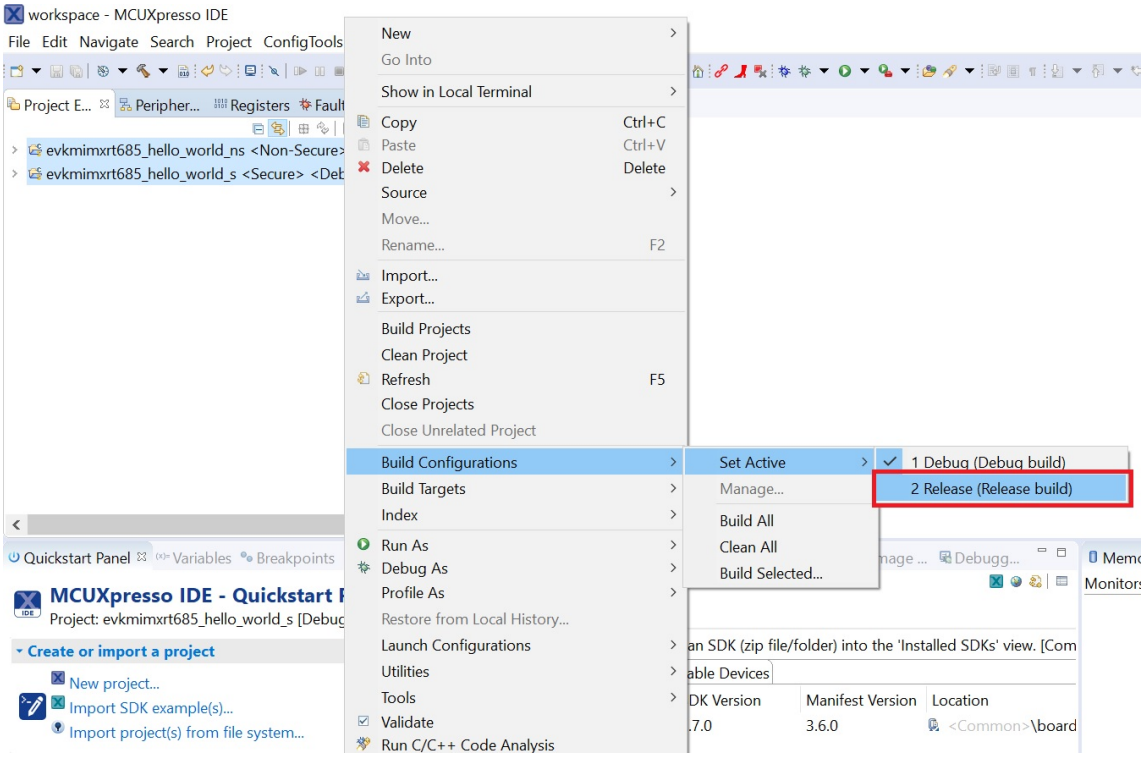
3. Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the evkmimxrt685_hello_world_s project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, **Debug, or Release**, by clicking the downward facing arrow next to the hammer icon, as shown in *Figure 3*. For this example, select the **Debug** target.



The project starts building after the build target is selected. It is requested to build the application for the secure project first, because the non-secure project must know the secure project since CMSE library when running the linker. It is not possible to finish the non-secure project linker when the secure project since CMSE library is not ready.

Note: When the **Release** build is requested, it is necessary to change the build configuration of both the secure and non-secure application projects first. To do this, select both projects in the Project Explorer view by clicking to select the first project, then using shift-click or control-click to select the second project. Right click in the Project Explorer view to

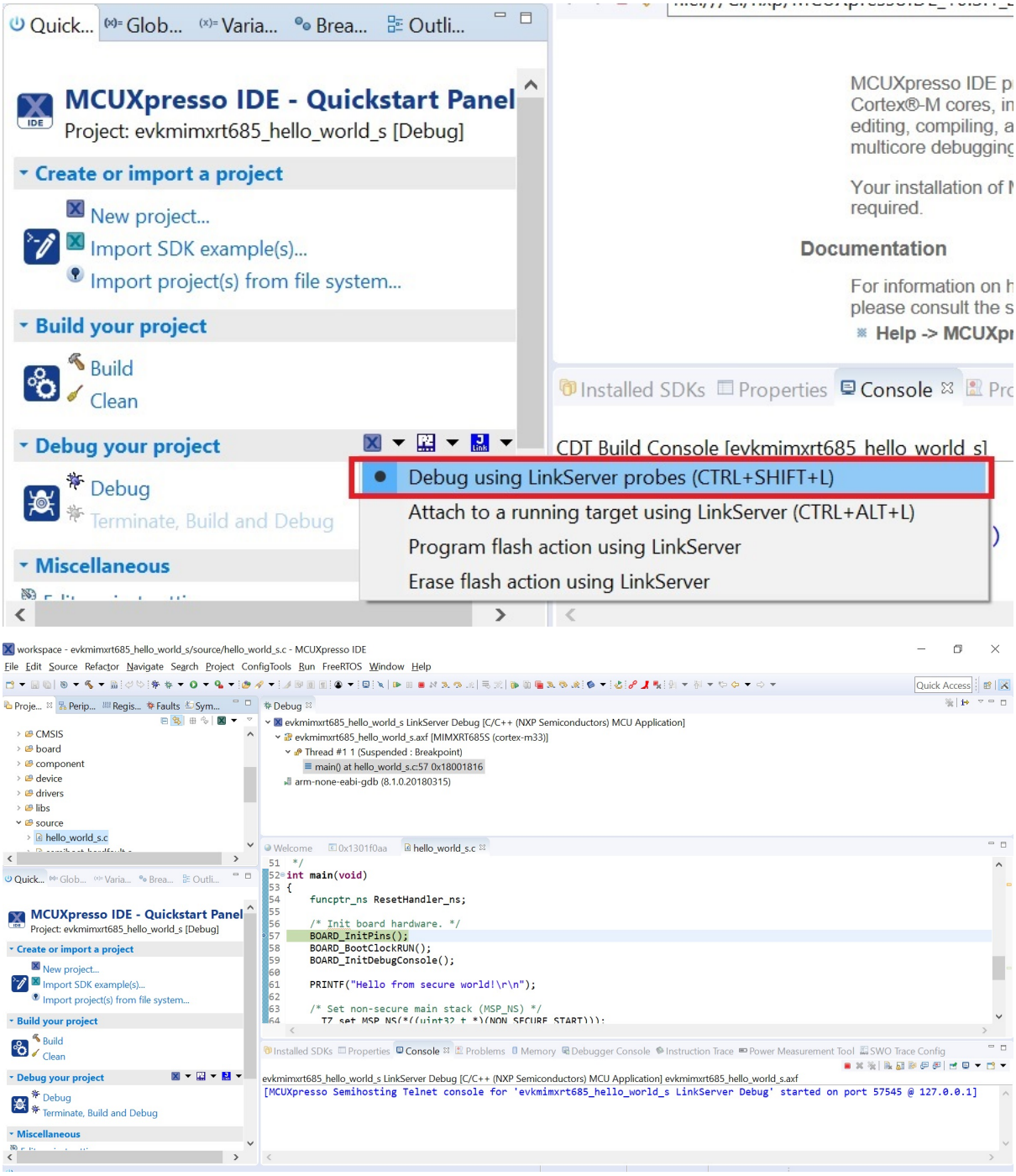
display the context-sensitive menu and select **Build Configurations** > **Set Active** >**Release**. This is also possible by using the menu item of **Project** > **Build Configuration** >**Set Active** >**Release**. After switching to the **Release** build configuration. Build the application for the secure project first.



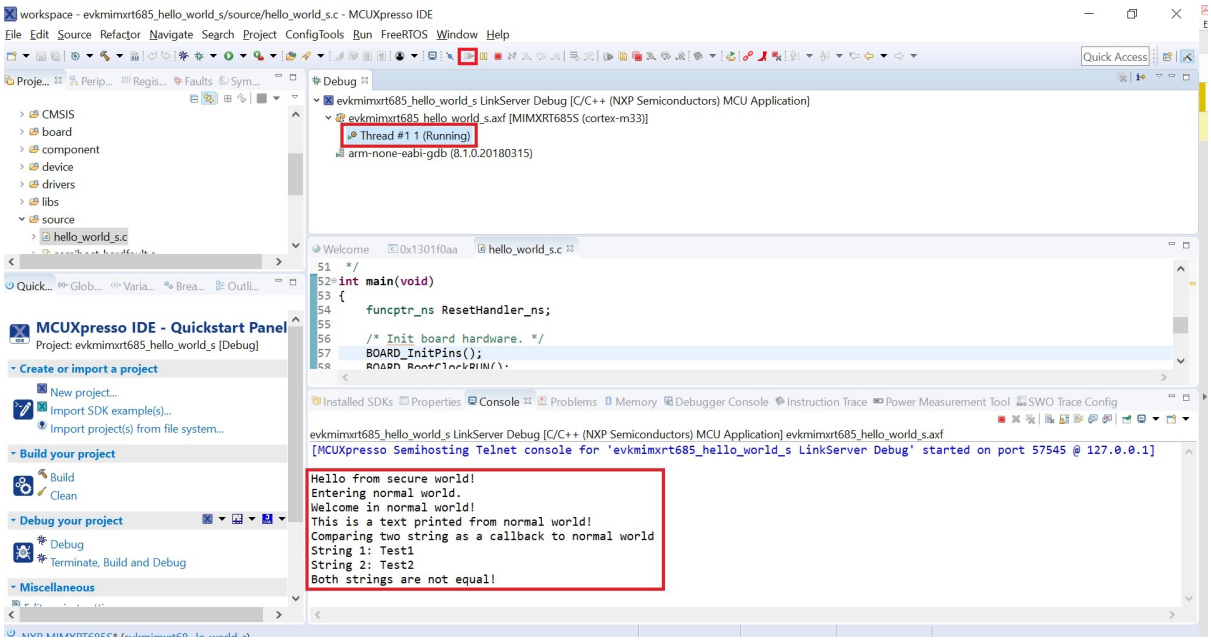
Parent topic:*Run a demo using MCUXpresso IDE*

Run a TrustZone example application To download and run the application, perform all steps as described in *Section 3.3, "Run an example application"*. These steps are common for single core, and TrustZone applications, ensuring evkmimxrt685_hello_world_s is selected for debugging.

In the Quickstart Panel, click **Debug evkmimxrt685_hello_world_s [Debug]** to launch the second debug session.



Now, the TrustZone sesions should be opened. Click **Resume**. The hello_world TrustZone application then starts running, and the secure application starts the non-secure application during runtime.



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 8.40.2 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* (document ID: MCUXSDKRN).

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
   ```

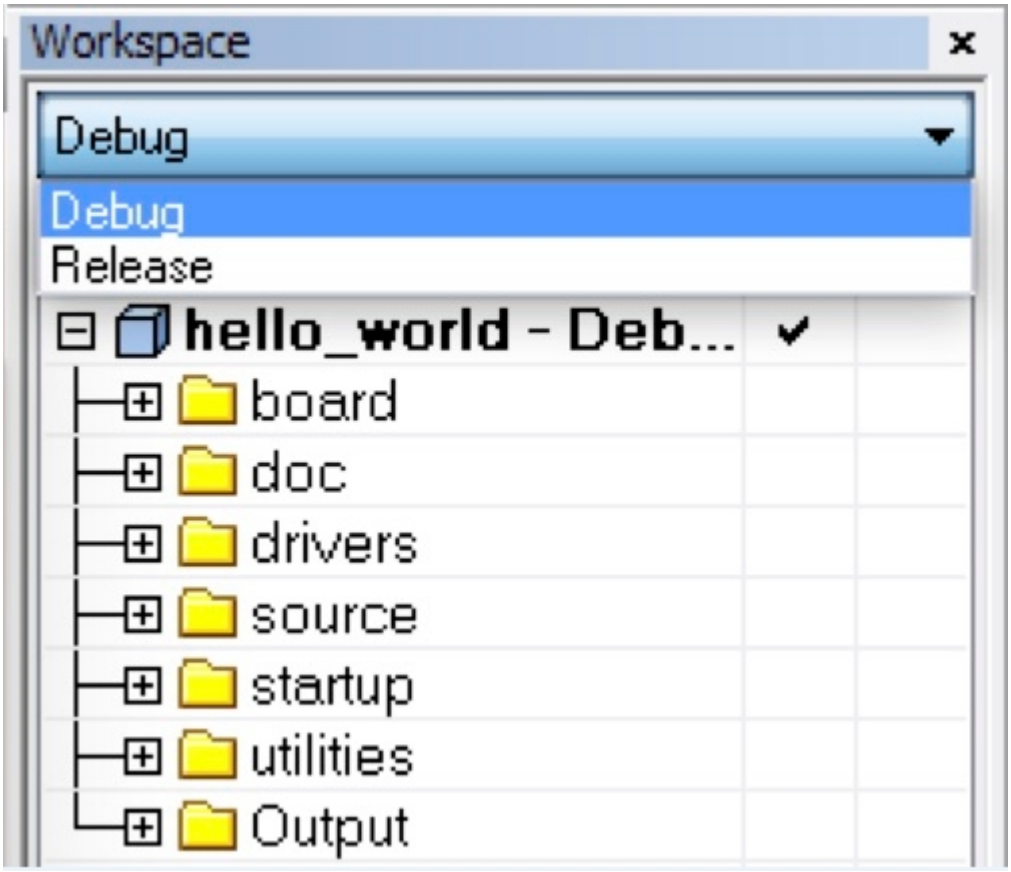
   Using the MIMXRT685-EVK hardware platform as an example, the hello_world workspace is located in:

   ```
   <install_dir>/boards/k32w061dk6/demo_apps/hello_world/iar/hello_world.eww
   ```

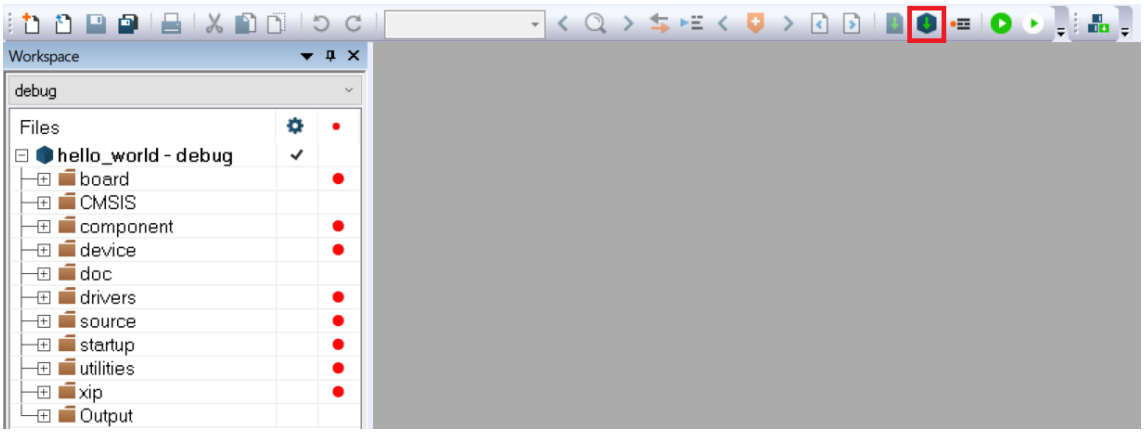
   Other example applications may have additional folders in their path.

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*.



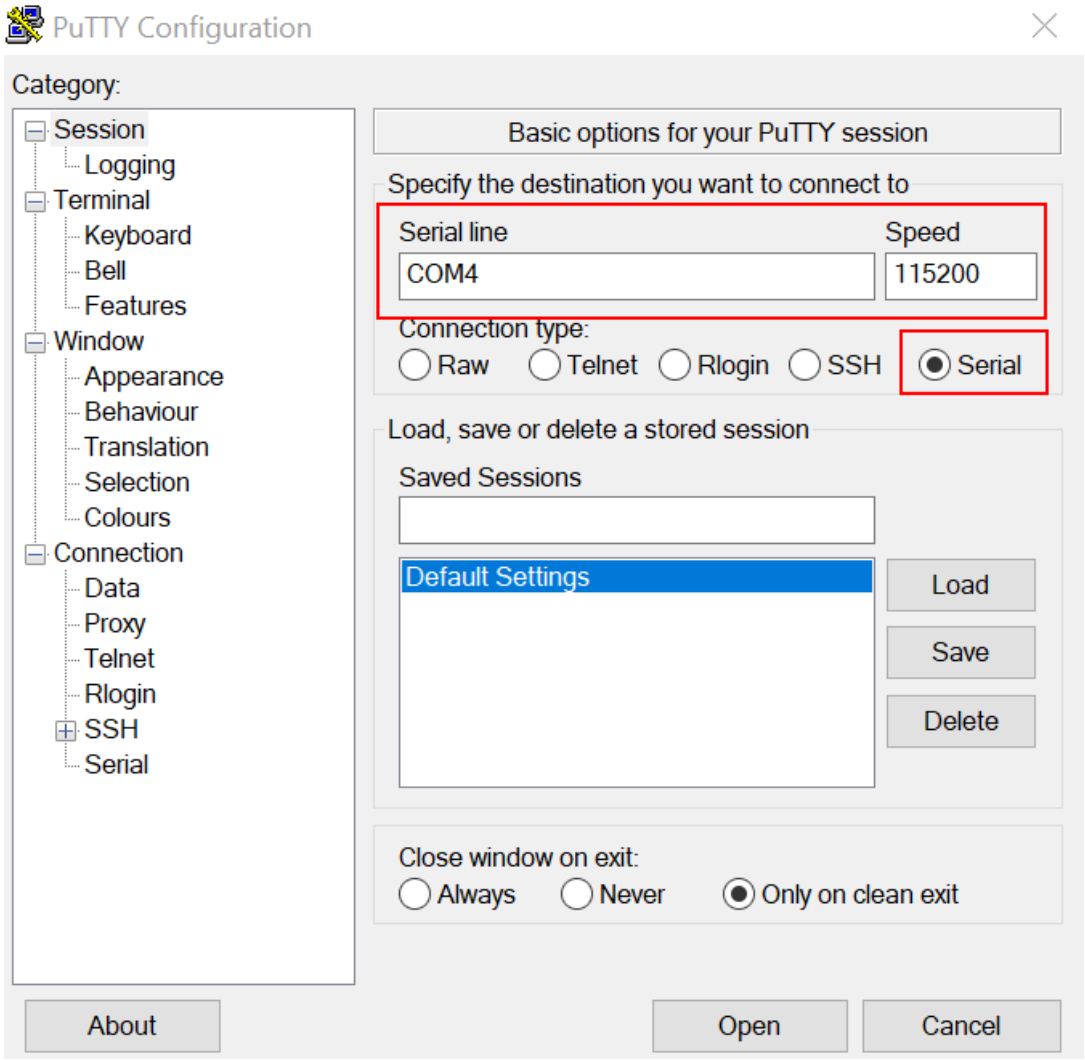
4. The build completes without errors.

Parent topic:*Run a demo application using IAR*

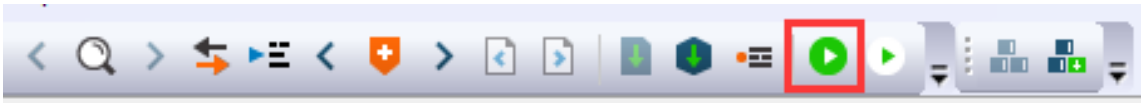
Run an example application To download and run the application, perform these steps:

1. See the table in *Default debug interfaces* to determine the debug interface that comes loaded on your specific hardware platform.
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:

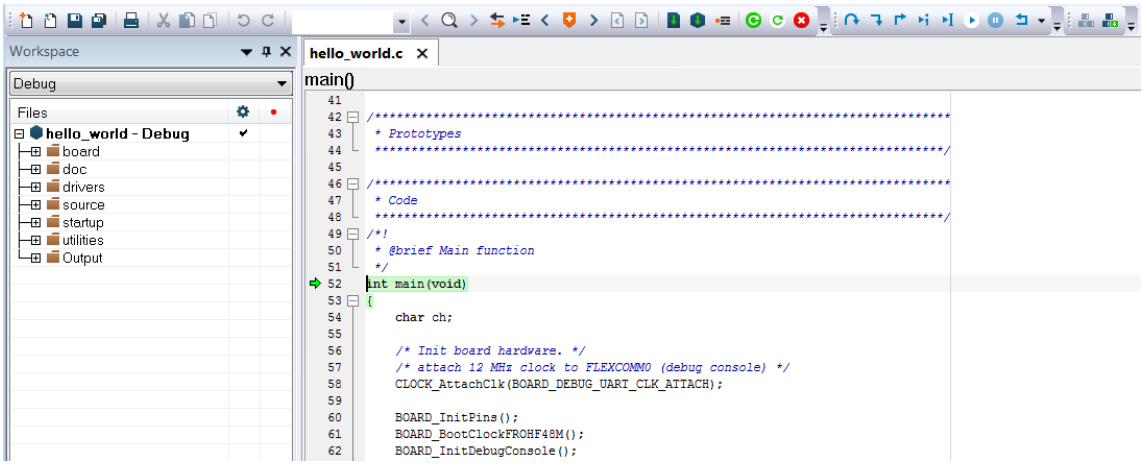
1. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
2. No parity
3. 8 data bits
4. 1 stop bit



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

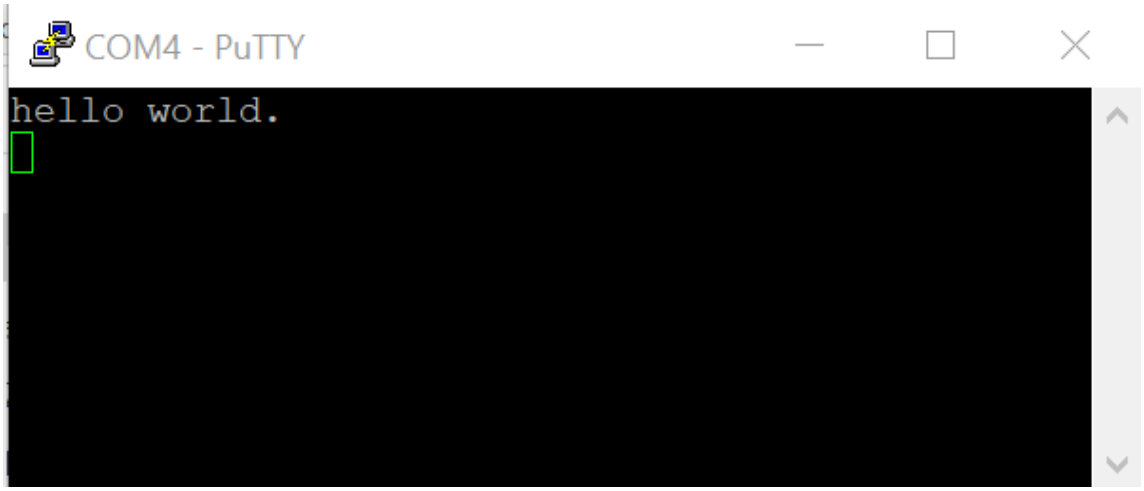


5. The application is then downloaded to the target and automatically runs to the main() function.



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 a TrustZone example application This section describes the particular steps that must be done in order to build and run a TrustZone application. The demo applications workspace files are located in this folder:

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/[<core_type>]/iar/
↪<application_name>_ns/iar
```

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/[<core_type>]/iar/
↪<application_name>_s/iar
```

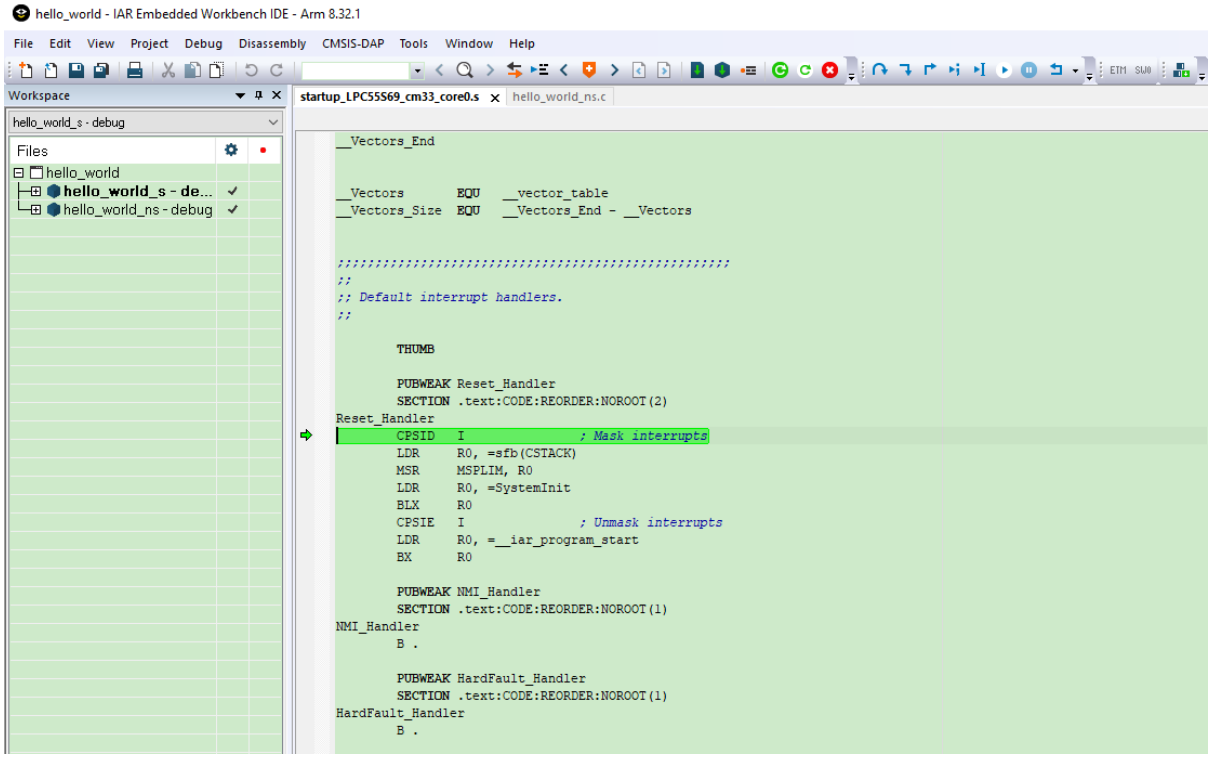
Begin with a simple TrustZone version of the Hello World application. The TrustZone Hello World IAR workspaces are located in this folder:

```
<install_dir>/boards/evkmimxrt685/trustzone_examples/hello_world/hello_world_s/iar/hello_world_s.
↪eww
```

```
<install_dir>/boards/evkmimxrt685/trustzone_examples/hello_world/hello_world_s/iar/hello_world.eww
```

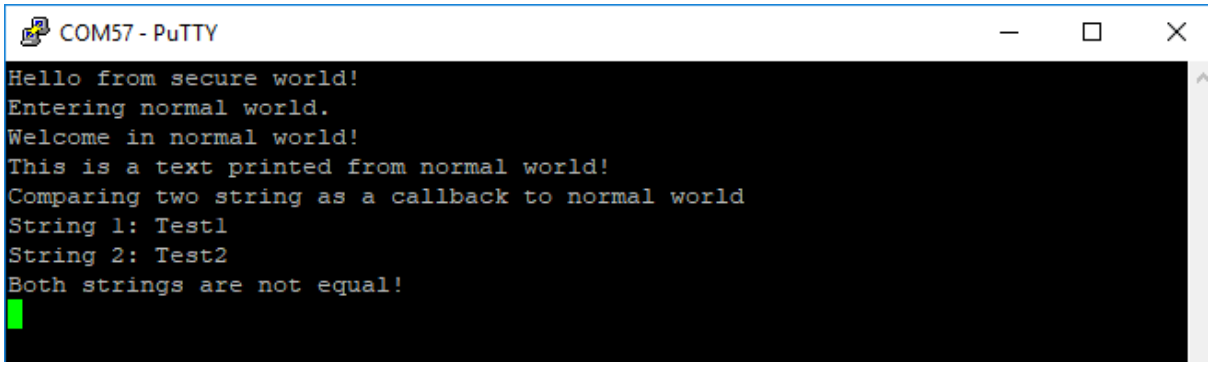
This project hello_world.eww contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because the non-secure project must know the secure project, since the CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project since CMSE library is not ready.

Run a TrustZone example application The secure project is configured to download both secure and non-secure output files, so debugging can be fully managed from the secure project. To download and run the TrustZone application, switch to the secure application project and perform steps 1 – 4 as described in *Section 4.2, Run an example application*. These steps are common for both single core, and TrustZone applications in IAR. After clicking **Download and Debug**, both the secure and non-secure images are loaded into the device memory, and the secure application is executed. It stops at the Rest_Hander function.

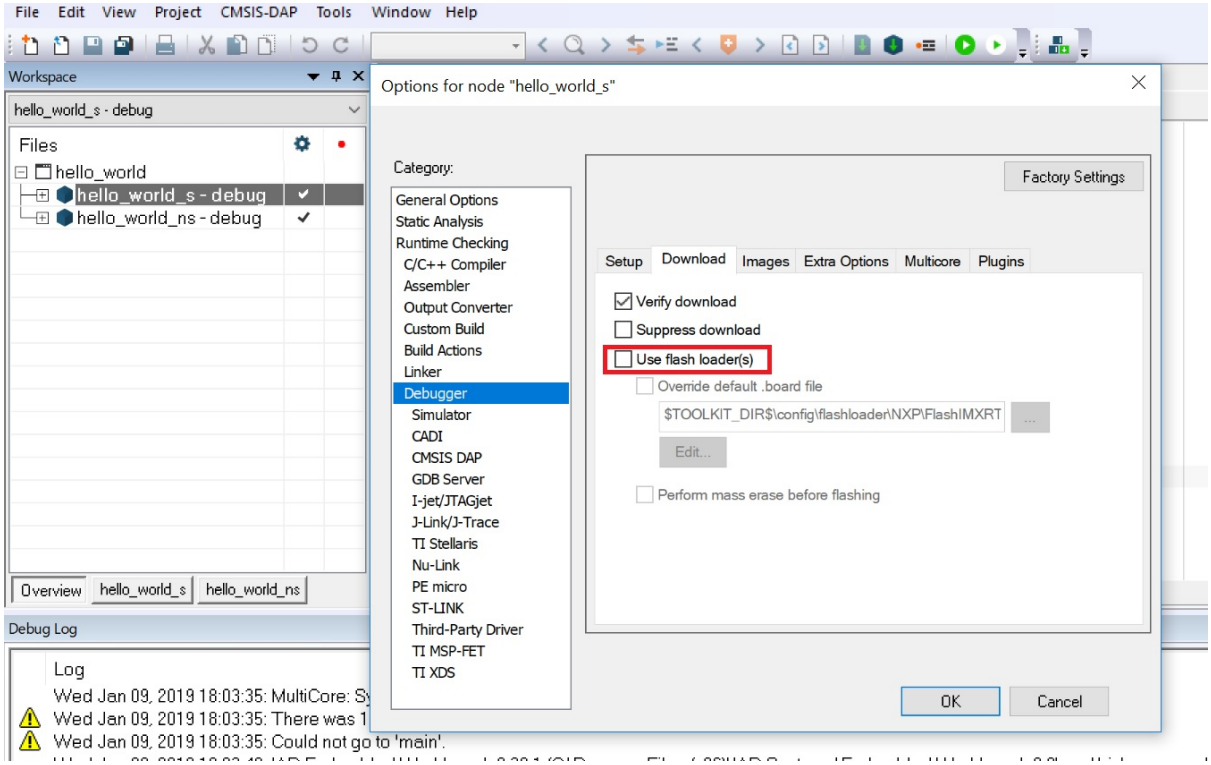


Run the code by clicking **Go** to start the application.

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



Note: If the application is running in RAM (debug/release build target), in **Options****>****Debugger** > **Download** tab, disable **Use flash loader(s)**. This can avoid the _ns download issue on i.MXRT600.



IAR RAM debugging notes In the SDK examples, there are many build configurations. For example, debug, release, flash_debug, and flash_release. The debug/release configuration demonstrates the application running in RAM, while flash_debug/flash_release runs in the flash (XIP).

XIP debugging is clean with the "**boot from flash**" setting on the board because the debugging has the same sequence as a normal boot.

The debugging steps are as follows:

1. The debugger downloads the image to the flash.
2. The debugger resets the device and boots from ROM.
3. The debugger halts the device and waits for the command from the user.

However, with RAM debugging with the *boot from flash* setting on the board, the situation is complicated.

1. The debugger downloads the image to the RAM.
2. The debugger resets the device and boots it from ROM.

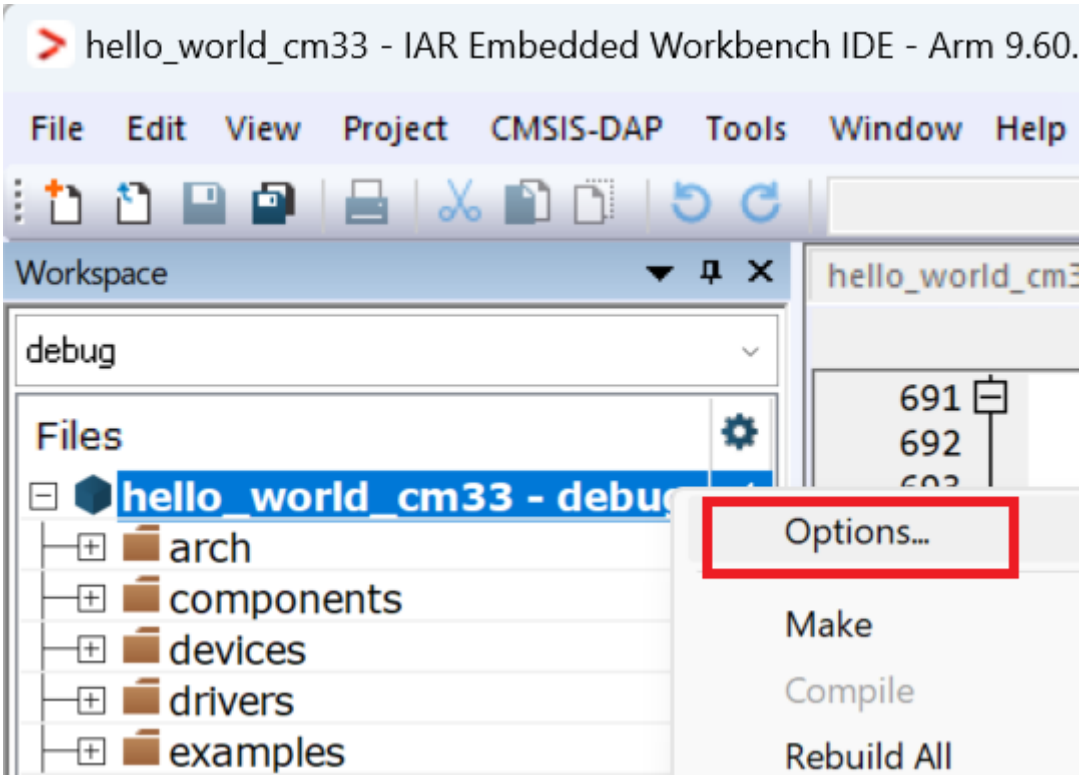
   Note: The image downloaded for debugging and the image running from the flash are different. The images are different because the image in flash might be written by the previous debugging.
3. The debugger halts the device and sets the PC register value to the entry address of the downloaded image.

The problem here is that step 2 might overwrite the RAM written in step 1. The other issue is that the hardware status is undetermined when PC is set to the start of the RAM application.

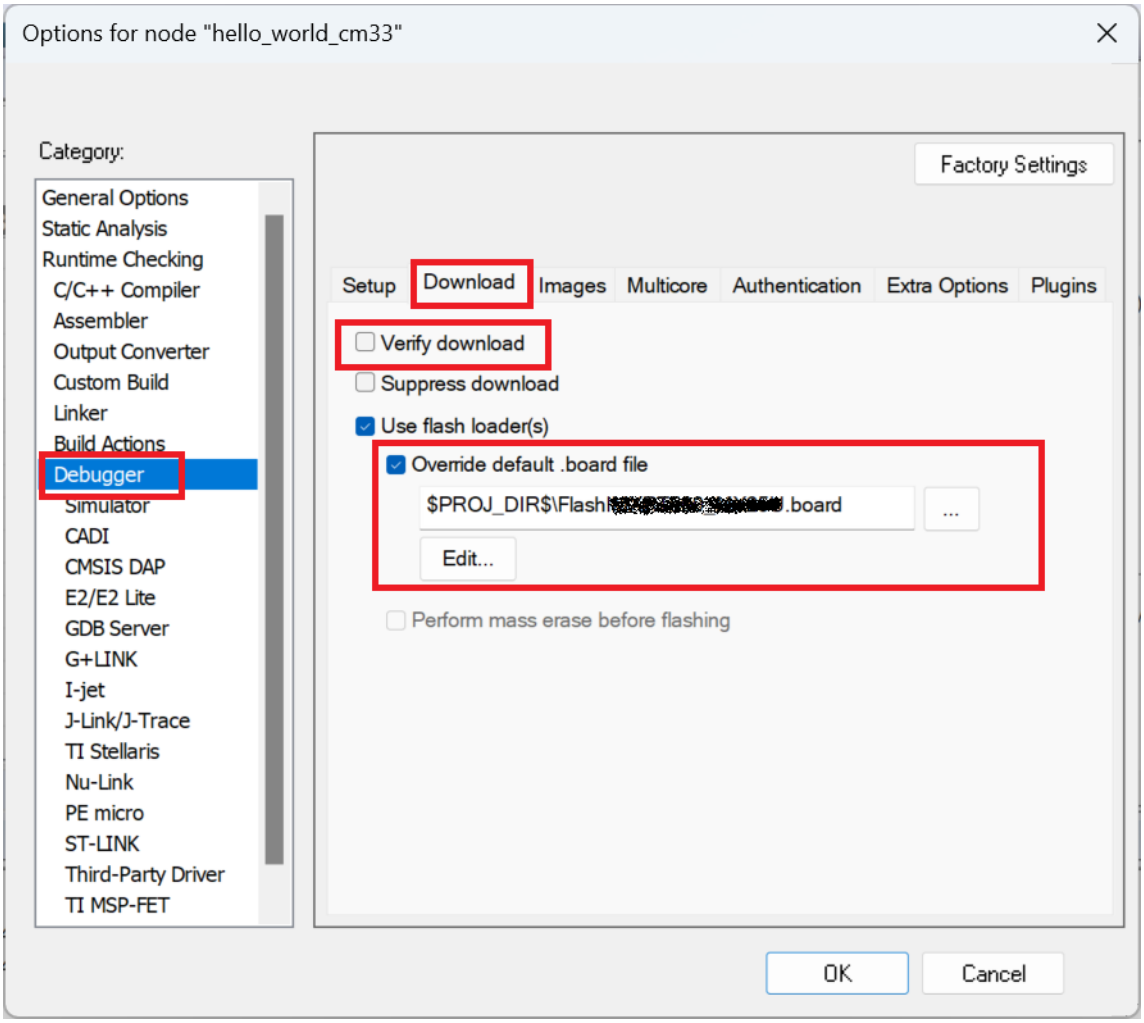
To avoid the RAM debugging issue, MCUX IDE/armgcc/MDK provides a plain load feature. To get a clean debugging status, the RAM image is downloaded to the flash.

IAR needs a special setting to achieve the plain load feature as follows:

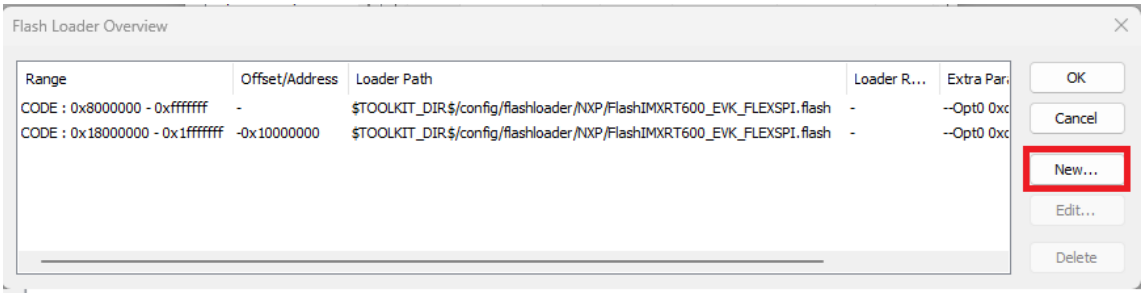
1. Right-click the project and select **Options**.



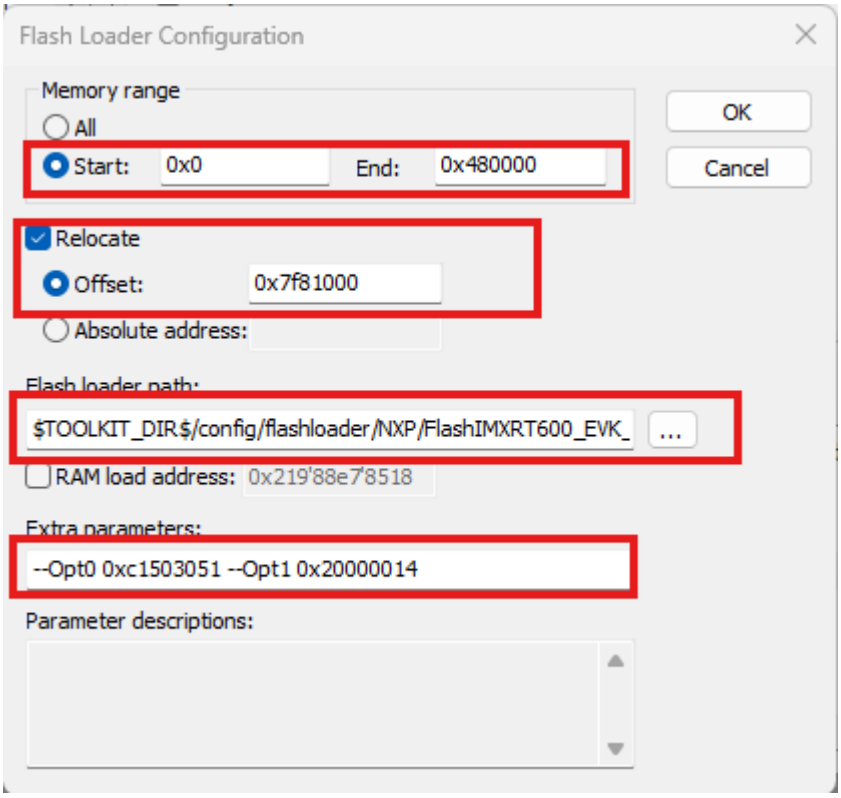
2. Select **Debugger** > **Download** in the **Options for node** <***project_name***> dialog box.
3. Clear the **Verify download** checkbox because the download address is different from the real address.
4. Select the **Override default.board file** option.
5. Copy the default *.board file to the IAR workspace folder.
6. Change the *.board file path.
7. Click the **Edit** button.



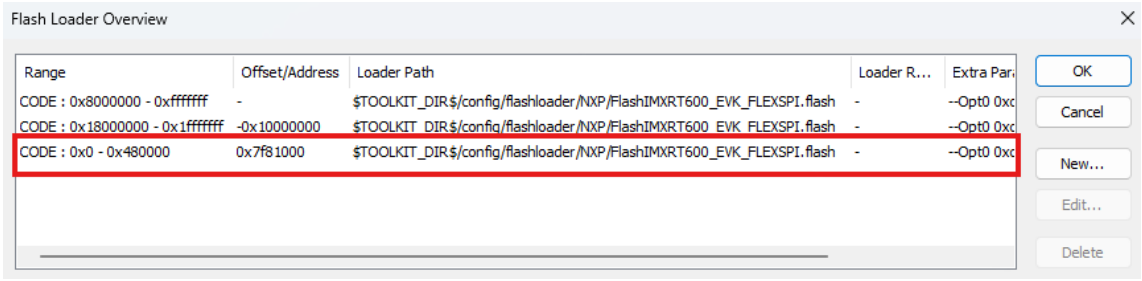
8. Create a range for RAM.



9. Set the memory range, relocate offset, and flashloader path.



The new RAM range appears in the flashloader overview.



10. Follow the instructions in Run an example application and now it is safe to debug the RAM application. As a result, the downloaded RAM application can also boot on reset.

Parent topic:*Run a demo application using IAR*

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.

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.

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 MIMXRT685S CMSIS pack.

1. Download the MIMXRT685S pack.
2. After downloading the DFP, double click to install it.

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

Build an example application

1. Open the desired example application workspace in:

   ```
   <install_dir>/boards/<board_name>/*<example\_type\>*/<application_name>/mdk
   ```

   The workspace file is named as <demo_name>.uvmpw. For this specific example, the actual path is:

   ```
   <install_dir>/boards/evkmimxrt685s/demo_apps/hello_world/mdk/hello_world.uvmpw
   ```

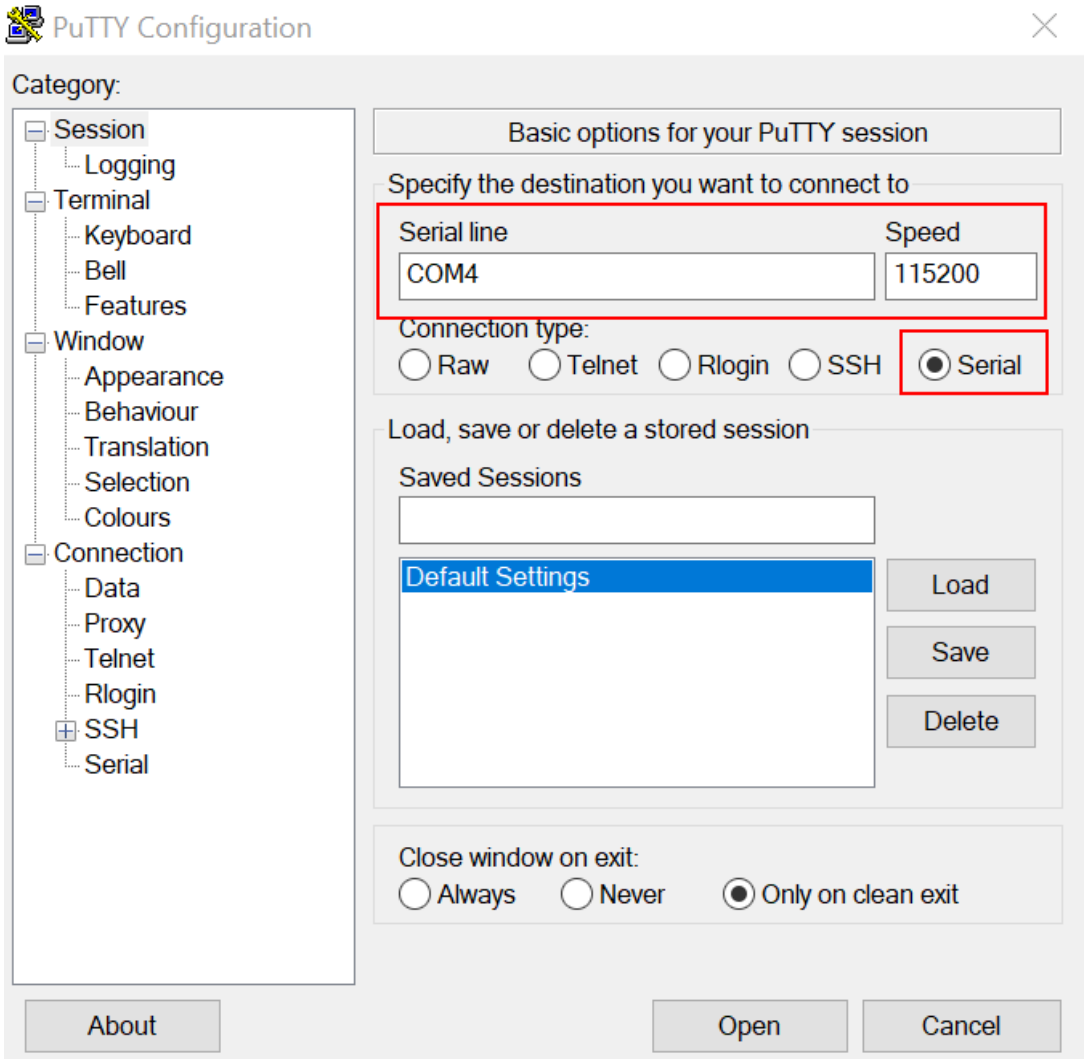
2. To build the demo project, select **Rebuild**, highlighted in red.

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. Reference the table in *Default debug interfaces* to determine the debug interface that comes loaded on your specific hardware platform.
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 serial port number (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

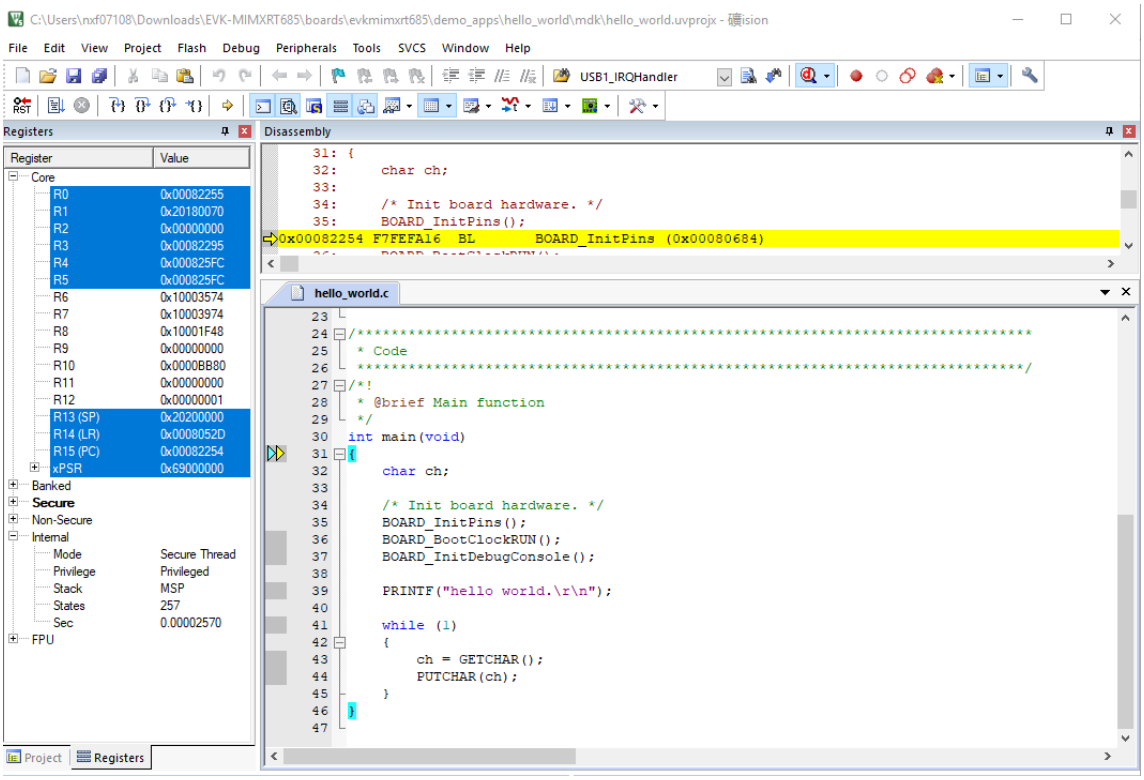


4. 1 stop bit

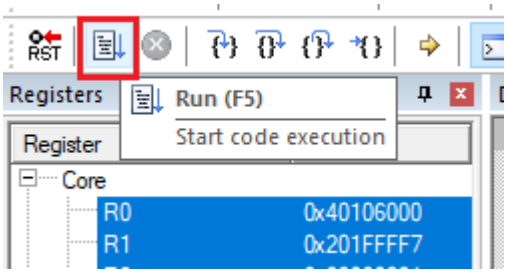
4. To debug the application, click **load** (or press the F8 key). Then, click the **Start/Stop Debug Session** button, highlighted in red in Figure 2. If using **J-Link** as the debugger, click **Project option** >**Debug** >**Settings** >**Debug** >**Port**, and select **SW**.

   Note: When debugging with jlink, it expects one jlinkscript file named JLinkSettings.JLinkScript in the folder where the uVision project files are located. For details, see SEGGER Wiki. For the contents in this JlinkSettings.JLinkScript, use contents in evkmimxrt1020_sdram_init.jlinkscript.

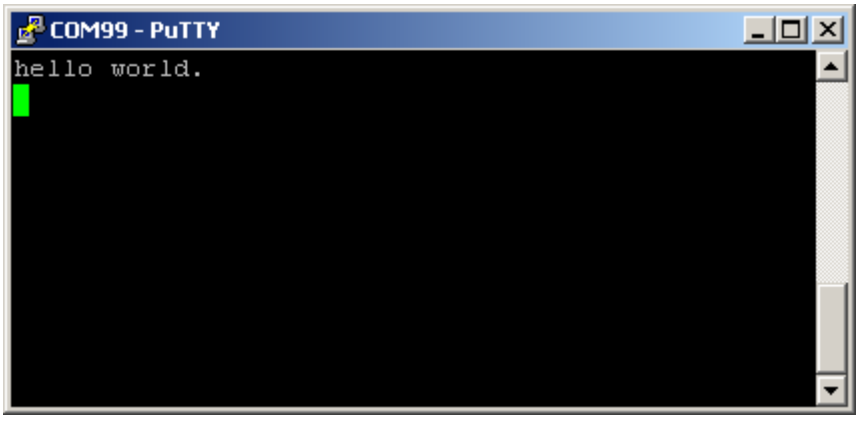
   Note: Make sure that the board is set to FlexSPI flash boot mode (ISP2: ISP1: ISP0 = ON, OFF, ON) before using Keil debug.



5. Run the code by clicking **Run** to start the application, as shown in Figure 3.



The hello_world application is now running and a banner is displayed on the terminal, as shown in Figure 4. If this is not true, check your terminal settings and connections.



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

Build a TrustZone example application This section describes the particular steps that must be done in order to build and run a TrustZone application. The demo applications workspace files are located in this folder:

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<application_name>_ns/
↪mdk
```

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<application_name>_s/
↪mdk
```

Begin with a simple TrustZone version of the Hello World application. The TrustZone Hello World Keil MSDK/µVision workspaces are located in this folder:

```
<install_dir>/boards/evkmimxrt595/trustzone_examples/hello_world/hello_world_ns/mdk/hello_world_ns.
↪uvmpw
```

```
<install_dir>/boards/evkmimxrt595/trustzone_examples/hello_world/hello_world_s/mdk/hello_world_s.
↪uvmpw
```

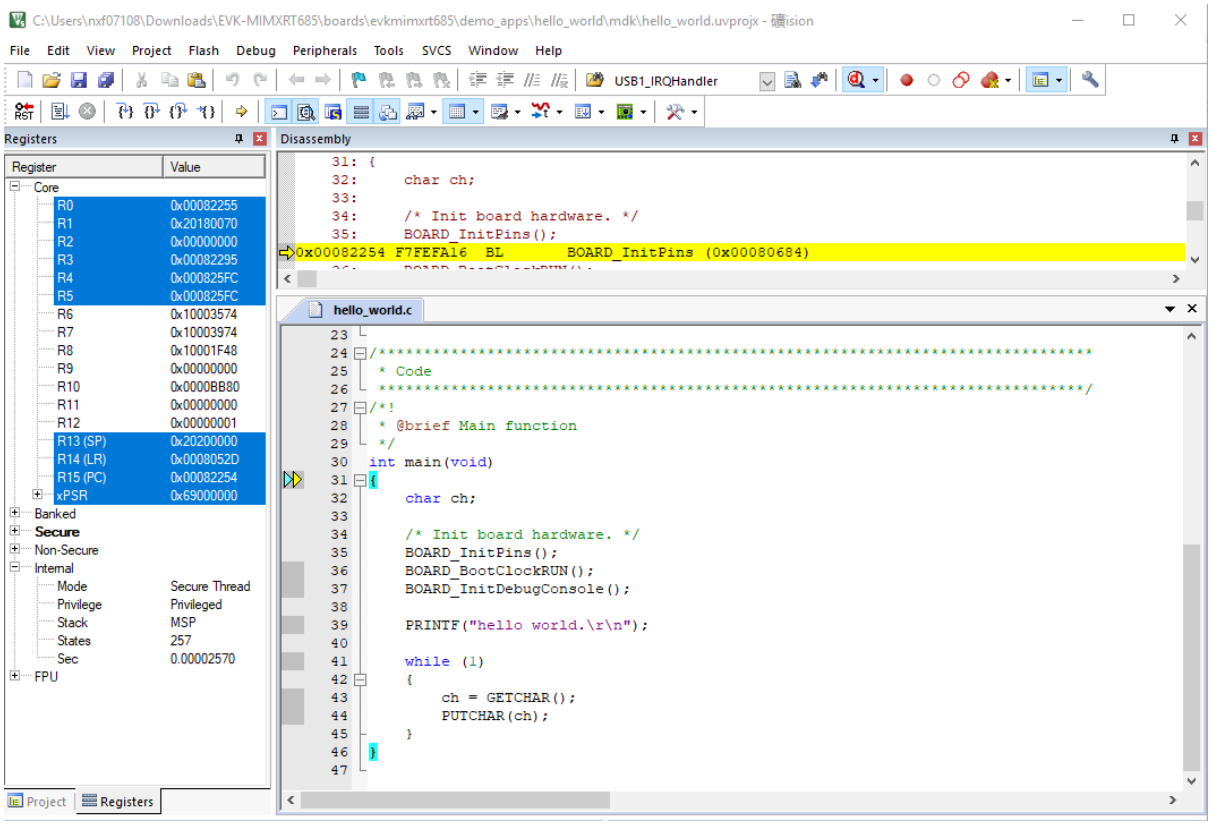
```
<install_dir>/boards/evkmimxrt595/trustzone_examples/hello_world/hello_world_s/mdk/hello_world.
↪uvmpw
```

This project hello_world.uvmpw contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another.

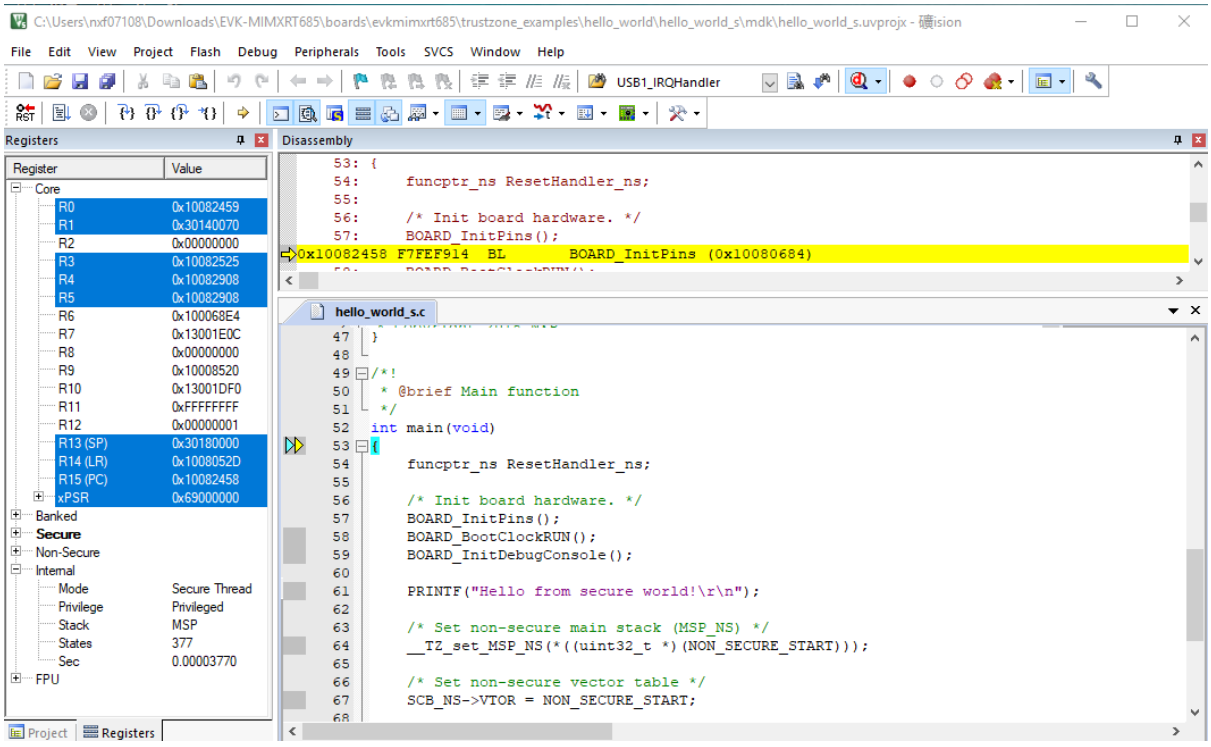
Build both applications separately by clicking **Rebuild**. It is requested to build the application for the secure project first, because the non-secure project must know the secure project since CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project because CMSE library is not ready.

Run a TrustZone example application The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

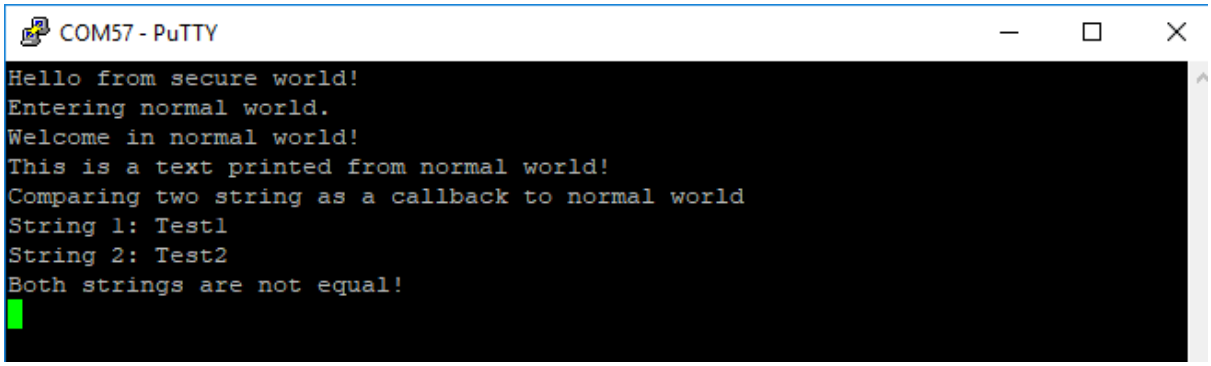
To download and run the TrustZone application, switch to the secure application project and perform steps as described in Run a TrustZone example application. These steps are common for single core, dual-core, and TrustZone applications in µVision. After clicking **Download and Debug**, both the secure and non-secure images are loaded into the device flash memory, and the secure application is executed. It stops at the main() function.



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



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



Run a demo using the prebuilt binary

This section describes the steps to write a prebuilt binary in the SDK package to the external Flash and boot from the external Flash. The hello_world demo application is used as an example.

Note: The prebuilt binaries are provided for quick evaluation of the SDK and the board. Only parts of the examples are provided with prebuilt binary such as examples under boards/<board_name>/demo_apps and boards/<board_name>/usb_examples.

Identify the load address of the binary Several toolchains are supported by the MCUXpresso SDK, usually the prebuilt binaries are built from armgcc flash_release targets using Arm GCC. If there is no flash_release target, then release target is used.

Take hello_world demo application as an example, the hello_world.bin is located in:

```
<install_dir>/boards/<board_name>/<demo_apps>/hello_world/
```

And it was built from:

```
<install_dir>/boards/<board_name>/<demo_apps>/hello_world/armgcc/build_flash_release.sh
```

The load address can be identified by the linker file in the same folder. Such as <device_name>_flash.ld.

```
MEMORY
{
  m_flash_config        (RX)  : ORIGIN = 0x08000400, LENGTH = 0x00000200
  m_interrupts          (RX)  : ORIGIN = 0x08001000, LENGTH = 0x00000130
  m_text                (RX)  : ORIGIN = 0x08001130, LENGTH = 0x001FEED0
  m_data                (RW)  : ORIGIN = 0x20080000, LENGTH = 0x00180000
  m_usb_sram            (RW)  : ORIGIN = 0x40140000, LENGTH = 0x00004000
```

So, the load address for hello_world.bin is 0x08000400.

Write the binary to external flash Take J-Link Commander as an example to write the hello_world.bin to external flash.

Run the J-Link Commander, run the following command to connect to the core.

```
J-Link>connect
J-Link>?
```

Select the correct device from the pop-up window **Target device settings**.

Specify targets interface as SWD and the target interface speed as per the guide in the command window. The **Cortex-M33 identified** appears when it is connected successfully.

MCUXpresso IDE New Project Wizard

MCUXpresso IDE features a new project wizard. The wizard provides functionality for the user to create new projects from the installed SDKs (and from pre-installed part support). It offers user the flexibility to select and change multiple builds. The wizard also includes a library and provides source code options. The source code is organized as software components, categorized as drivers, utilities, and middleware.

To use the wizard, start the MCUXpresso IDE. This is located in the **QuickStart Panel** at the bottom left of the MCUXpresso IDE window. Select **New project**, as shown in *Figure 1*.

For more details and usage of new project wizard, see the *MCUXpresso_IDE_User_Guide.pdf* in the MCUXpresso IDE installation folder.

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 Cortex-A core debug console and the other is for Cortex M4.

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 will be 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
}
```

Default debug interfaces

The MCUXpresso SDK supports various hardware platforms that come loaded with various factory programmed debug interface configurations. *Table 1* lists the hardware platforms supported by the MCUXpresso SDK, their default debug interface, and any version information that helps differentiate a specific interface configuration.

Note: The *OpenSDA details* column in *Table 1* is not applicable to LPC.

Hardware platform	Default interface	OpenSDA details
EVK-MC56F83000	P&E Micro OSJTAG	N/A
EVK-MIMXRT595	CMSIS-DAP	N/A
EVK-MIMXRT685	CMSIS-DAP	N/A
FRDM-K22F	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.1
FRDM-K28F	DAPLink	OpenSDA v2.1
FRDM-K32L2A4S	CMSIS-DAP	OpenSDA v2.1
FRDM-K32L2B	CMSIS-DAP	OpenSDA v2.1
FRDM-K32W042	CMSIS-DAP	N/A
FRDM-K64F	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.0
FRDM-K66F	J-Link OpenSDA	OpenSDA v2.1
FRDM-K82F	CMSIS-DAP	OpenSDA v2.1
FRDM-KE15Z	DAPLink	OpenSDA v2.1
FRDM-KE16Z	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.2
FRDM-KL02Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL03Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL25Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL26Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL27Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL28Z	P&E Micro OpenSDA	OpenSDA v2.1
FRDM-KL43Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL46Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL81Z	CMSIS-DAP	OpenSDA v2.0

continues on next page

Table 1 – continued from previous page

Hardware platform	Default interface	OpenSDA details
FRDM-KL82Z	CMSIS-DAP	OpenSDA v2.0
FRDM-KV10Z	CMSIS-DAP	OpenSDA v2.1
FRDM-KV11Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KV31F	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KW24	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.1
FRDM-KW36	DAPLink	OpenSDA v2.2
FRDM-KW41Z	CMSIS-DAP/DAPLink	OpenSDA v2.1 or greater
Hexiwear	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.0
HVP-KE18F	DAPLink	OpenSDA v2.2
HVP-KV46F150M	P&E Micro OpenSDA	OpenSDA v1
HVP-KV11Z75M	CMSIS-DAP	OpenSDA v2.1
HVP-KV58F	CMSIS-DAP	OpenSDA v2.1
HVP-KV31F120M	P&E Micro OpenSDA	OpenSDA v1
JN5189DK6	CMSIS-DAP	N/A
LPC54018 IoT Module	N/A	N/A
LPCXpresso54018	CMSIS-DAP	N/A
LPCXpresso54102	CMSIS-DAP	N/A
LPCXpresso54114	CMSIS-DAP	N/A
LPCXpresso51U68	CMSIS-DAP	N/A
LPCXpresso54608	CMSIS-DAP	N/A
LPCXpresso54618	CMSIS-DAP	N/A
LPCXpresso54628	CMSIS-DAP	N/A
LPCXpresso54S018M	CMSIS-DAP	N/A
LPCXpresso55s16	CMSIS-DAP	N/A
LPCXpresso55s28	CMSIS-DAP	N/A
LPCXpresso55s69	CMSIS-DAP	N/A
MAPS-KS22	J-Link OpenSDA	OpenSDA v2.0
MIMXRT1170-EVK	CMSIS-DAP	N/A
TWR-K21D50M	P&E Micro OSJTAG	N/AOpenSDA v2.0
TWR-K21F120M	P&E Micro OSJTAG	N/A
TWR-K22F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K24F120M	CMSIS-DAP/mbed	OpenSDA v2.1
TWR-K60D100M	P&E Micro OSJTAG	N/A
TWR-K64D120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K64F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K65D180M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K65D180M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV10Z32	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K80F150M	CMSIS-DAP	OpenSDA v2.1
TWR-K81F150M	CMSIS-DAP	OpenSDA v2.1
TWR-KE18F	DAPLink	OpenSDA v2.1
TWR-KL28Z72M	P&E Micro OpenSDA	OpenSDA v2.1
TWR-KL43Z48M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KL81Z72M	CMSIS-DAP	OpenSDA v2.0
TWR-KL82Z72M	CMSIS-DAP	OpenSDA v2.0
TWR-KM34Z75M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KM35Z75M	DAPLink	OpenSDA v2.2
TWR-KV10Z32	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV11Z75M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV31F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV46F150M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV58F220M	CMSIS-DAP	OpenSDA v2.1
TWR-KW24D512	P&E Micro OpenSDA	OpenSDA v1.0
USB-KW24D512	N/A External probe	N/A

continues on next page

Table 1 – continued from previous page

Hardware platform	Default interface	OpenSDA details
USB-KW41Z	CMSIS-DAP\DAPLink	OpenSDA v2.1 or greater

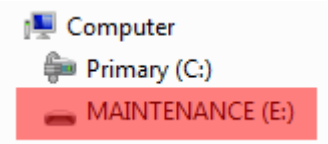
Updating debugger firmware

Updating OpenSDA firmware Any NXP hardware platform that comes with an OpenSDA-compatible debug interface has the ability to update the OpenSDA firmware. For reference, OpenSDA firmware files can be found at the links below:

- J-Link: Download appropriate image from www.segger.com/opensda.html. Choose the appropriate J-Link binary based on the table in Default debug interfaces. Any OpenSDA v1.0 interface should use the standard OpenSDA download (in other words, the one with no version). For OpenSDA 2.0 or 2.1, select the corresponding binary.
- P&E Micro: Downloading P&E Micro OpenSDA firmware images requires registration with P&E Micro (www.pemicro.com).

Perform the following steps to update the OpenSDA firmware on your board for Windows and Linux OS users:

1. Unplug the board's USB cable.
2. Press the **Reset** button on the board. While still holding the button, plug the USB cable back into the board.
3. When the board re-enumerates, it shows up as a disk drive called **MAINTENANCE**.



4. Drag and drop the new firmware image onto the MAINTENANCE drive.

   Note: If for any reason the firmware update fails, the board can always reenter maintenance mode by holding down **Reset** button and power cycling.

These steps show how to update the OpenSDA firmware on your board for Mac OS users.

1. Unplug the board's USB cable.
2. Press the **Reset** button of the board. While still holding the button, plug the USB cable back into the board.
3. For boards with OpenSDA v2.0 or v2.1, it shows up as a disk drive called **BOOTLOADER** in **Finder**. Boards with OpenSDA v1.0 may or may not show up depending on the bootloader version. If you see the drive in **Finder**, proceed to the next step. If you do not see the drive in Finder, use a PC with Windows OS 7 or an earlier version to either update the OpenSDA firmware, or update the OpenSDA bootloader to version 1.11 or later. The bootloader update instructions and image can be obtained from P&E Microcomputer website.
4. For OpenSDA v2.1 and OpenSDA v1.0 (with bootloader 1.11 or later) users, drag the new firmware image onto the BOOTLOADER drive in **Finder**.
5. For OpenSDA v2.0 users, type these commands in a Terminal window:

```
> sudo mount -u -w -o sync /Volumes/BOOTLOADER
> cp -X  <path to update file> /Volumes/BOOTLOADER
```

   Note: If for any reason the firmware update fails, the board can always reenter bootloader mode by holding down the **Reset** button and power cycling.

Updating LPCXpresso board firmware The LPCXpresso hardware platform comes with a CMSIS-DAP-compatible debug interface (known as LPC-Link2). This firmware in this debug interface may be updated using the host computer utility called LPCScrypt. 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 reprogram the debug probe firmware.

Note: If MCUXpresso IDE is used and the jumper making DFUlink is installed on the board (JP5 on some boards, but consult the board user manual or schematic for specific jumper number), LPC-Link2 debug probe boots to DFU mode, and MCUXpresso IDE automatically downloads the CMSIS-DAP firmware to the probe before flash memory programming (after clicking **Debug**). Using DFU mode ensures that most up-to-date/compatible firmware is used with MCUXpresso IDE.

NXP provides the LPCScrypt utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto LPC-Link2 or LPCXpresso boards. The utility can be downloaded from www.nxp.com/lpcutilities.

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in LPCScrypt user guide (www.nxp.com/lpcutilities, select **LPCScrypt**, and then the documentation tab).

1. Install the LPCScript utility.
2. Unplug the board's USB cable.
3. Make the DFU link (install the jumper labeled DFUlink).
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 LPCScrypt installation directory (<LPCScrypt install dir>).
    1. To program CMSIS-DAP debug firmware: <LPCScrypt install dir>/scripts/program_CMSIS
    2. To program J-Link debug firmware: <LPCScrypt install dir>/scripts/program_JLINK
6. Remove DFU link (remove the jumper installed in Step 3).
7. Repower the board by removing the USB cable and plugging it in again.

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.

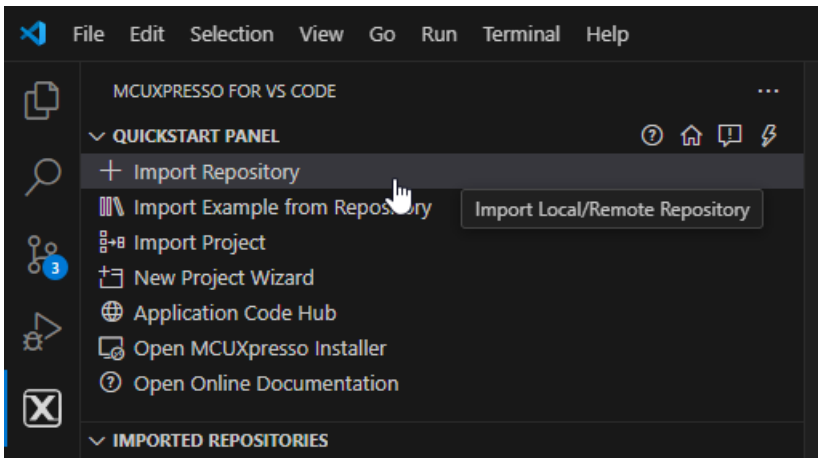
SDK Import and Setup

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.

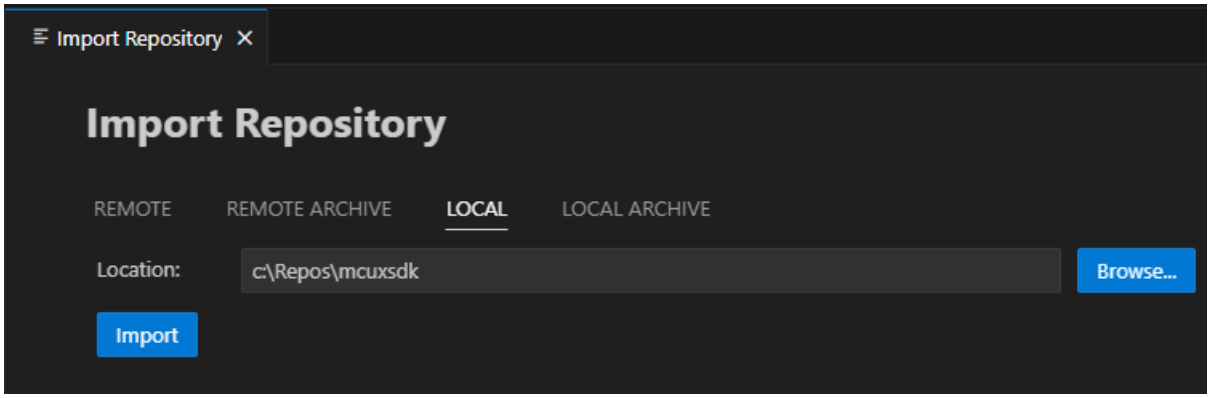
Import GitHub Repository SDK

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

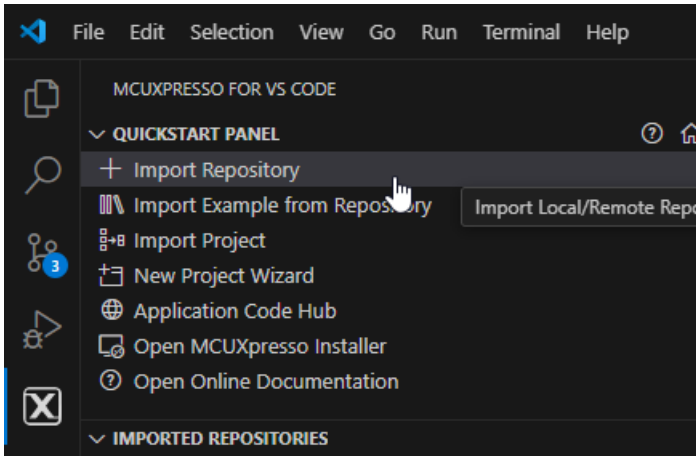


Note: You can import the SDK in several ways. Refer to the MCUXpresso for VS Code documentation for details.

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

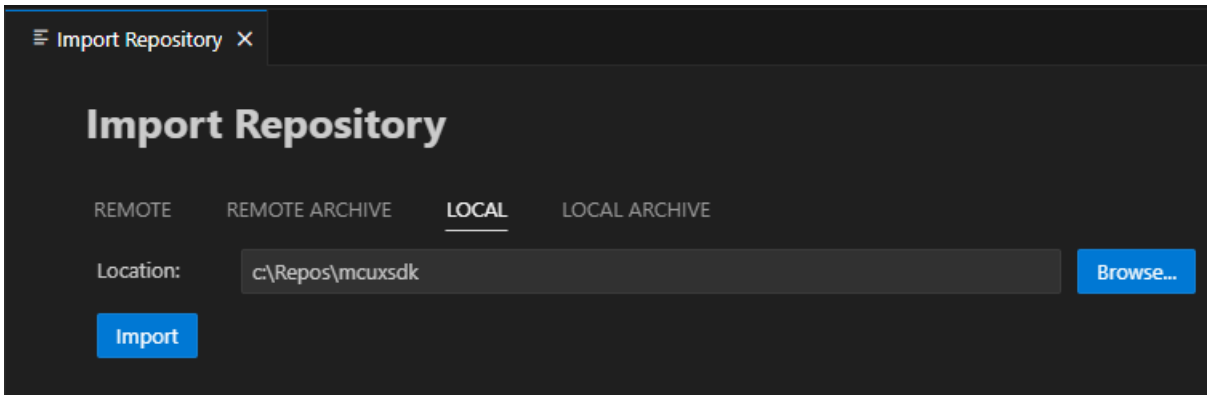


Import Repository-Layout SDK Package

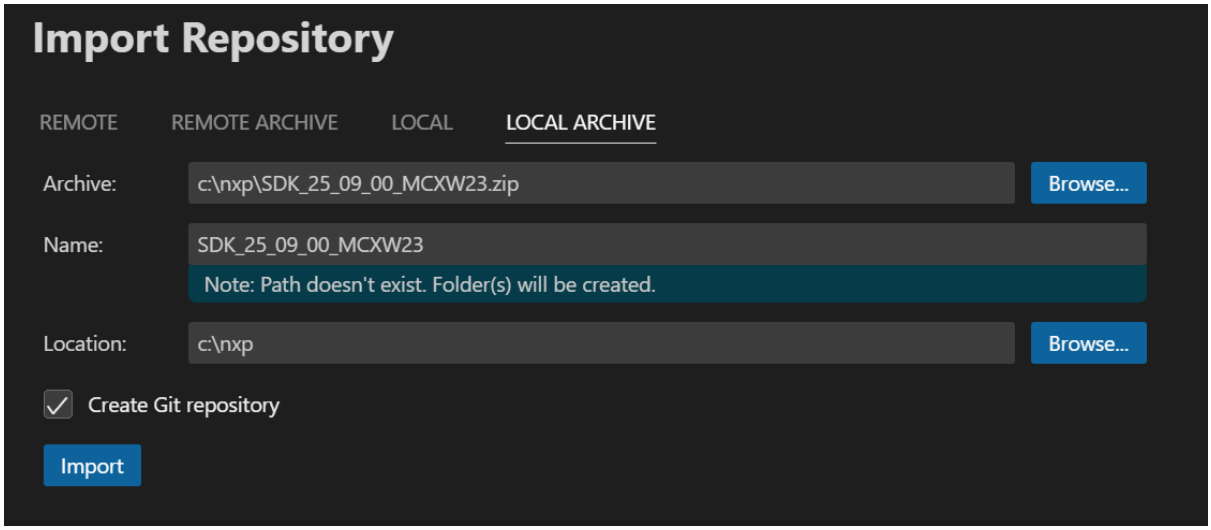


Click **Import Repository** from the **QUICKSTART PANEL**

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



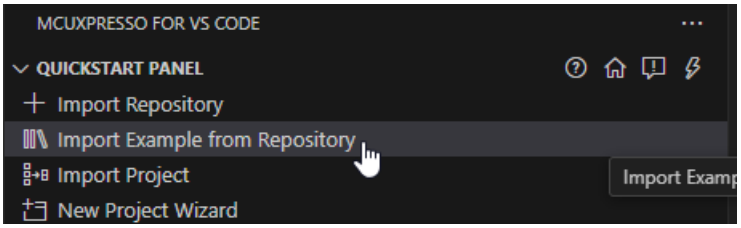
Else if the SDK is ZIP archive, select **Local Archive**, browse to the downloaded SDK ZIP file, fill the link of expect location, then click **Import**.



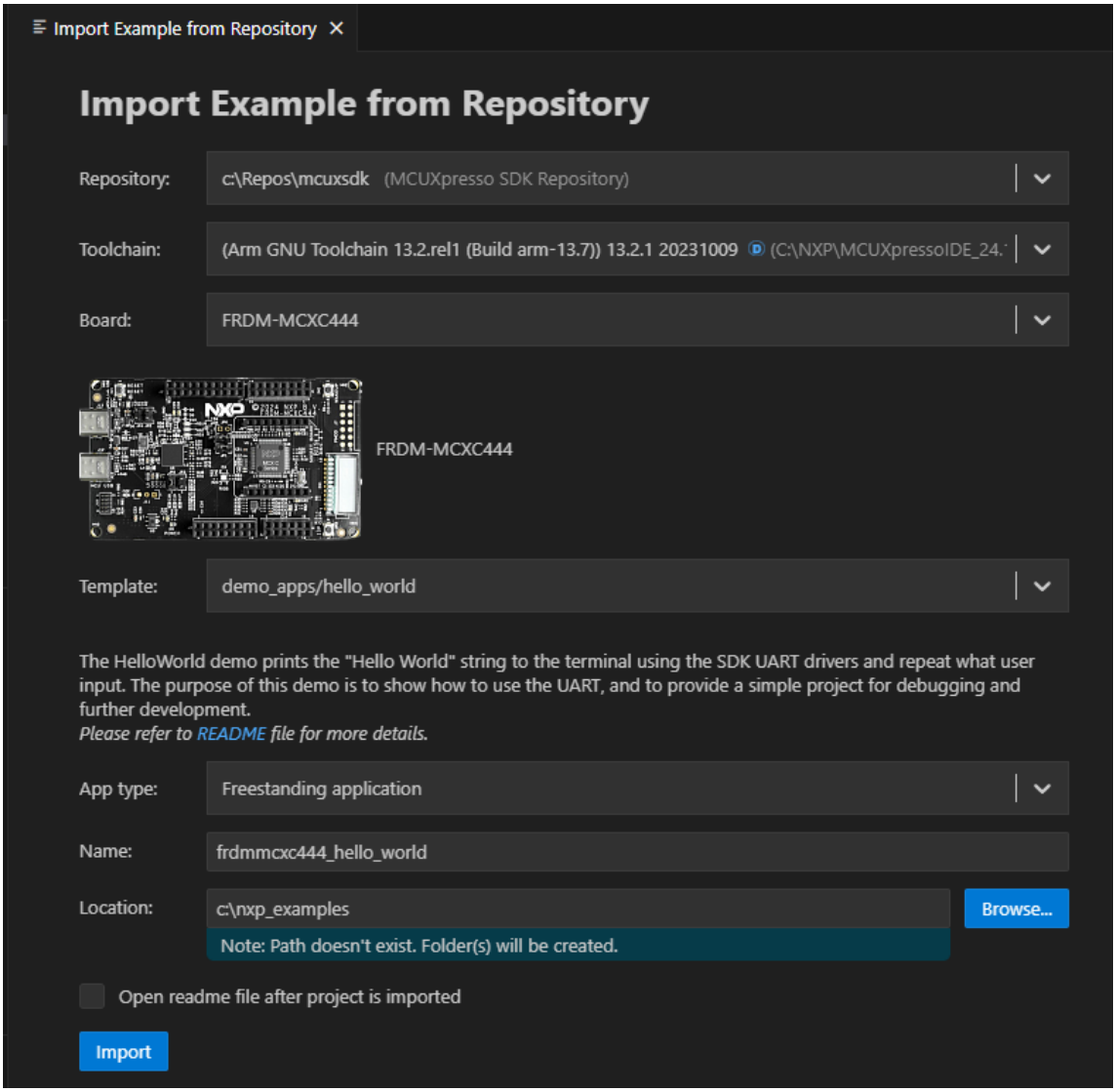
Building Example Applications

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

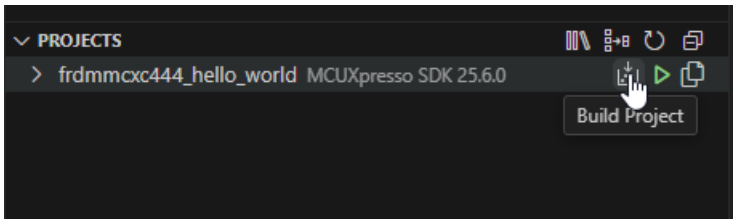
VS Code will prompt you to confirm if the imported files are trusted. Click **Yes** to proceed.

Building Projects

Build Process

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

3. Click the **Build Project** icon

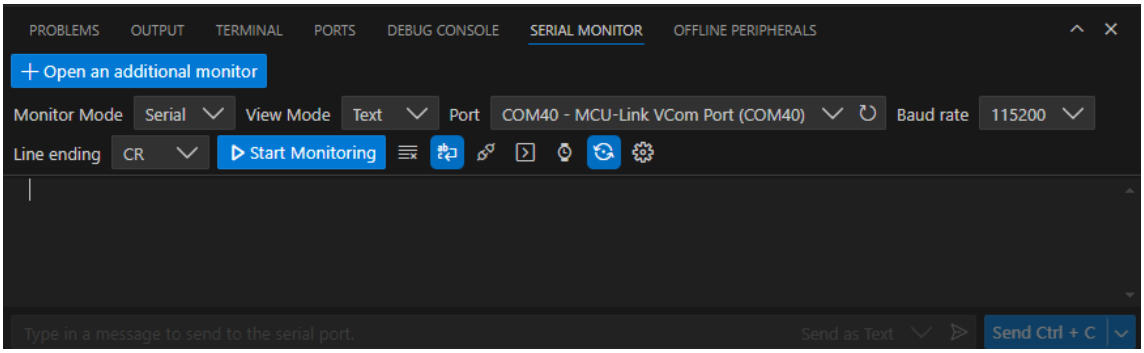


The integrated terminal will display build output at the bottom of the VS Code window.

Running and Debugging

Serial Monitor Setup

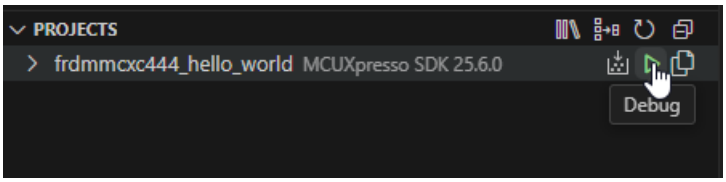
1. Open **Serial Monitor** from VS Code's integrated terminal



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



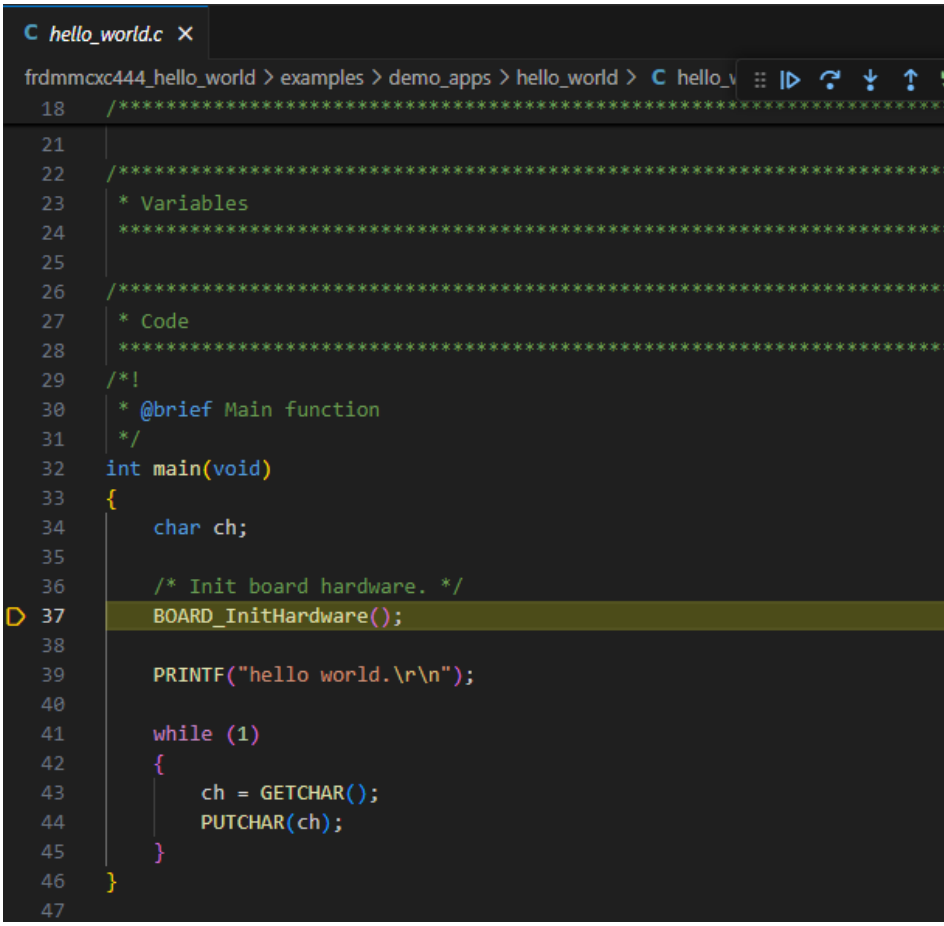
The debug session will begin with debug controls initially at the top of the interface.

Debug Controls

Use the debug controls to manage execution:

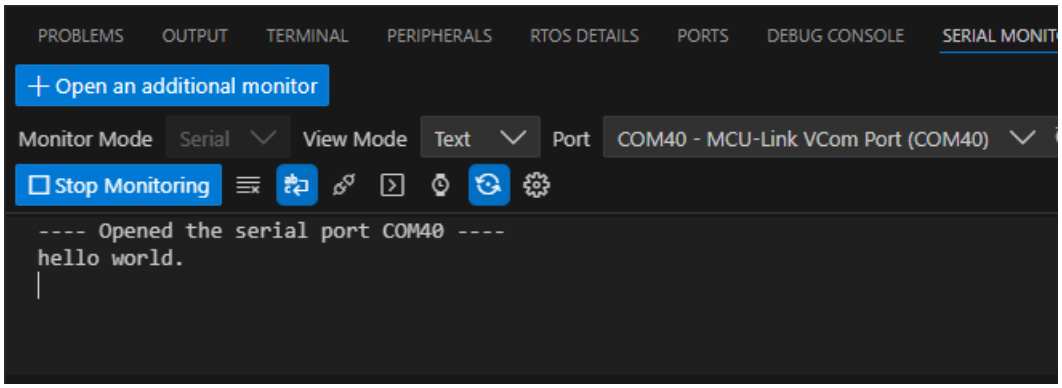
- **Continue**: Resume code execution
- **Step controls**: Navigate through code

- **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 RT600 DSP Build Configuration
- Install Xtensa On Chip Debugger Daemon
- Program LPC-Link2 with SEGGER J-Link
- 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/rt600-download-page/ 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 RT600 DSP Build Configuration. **Note:** NXP recommends version **10.1.11** of the Xtensa Xplorer IDE and tools for use with the RT600 DSP.

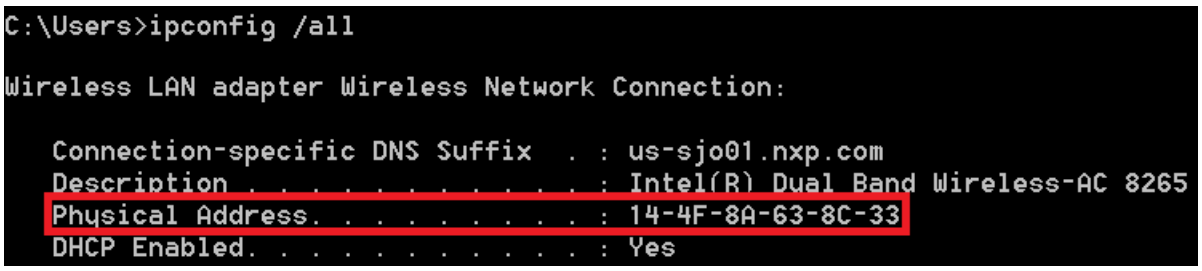
Install License Key Xtensa development tools use FLEXlm for license management. FLEXlm licensing is required for tools such as the Xtensa Xplorer IDE, Xtensa C and C++ compiler, and Instruction Set Simulator (ISS).

Currently RT600 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
```

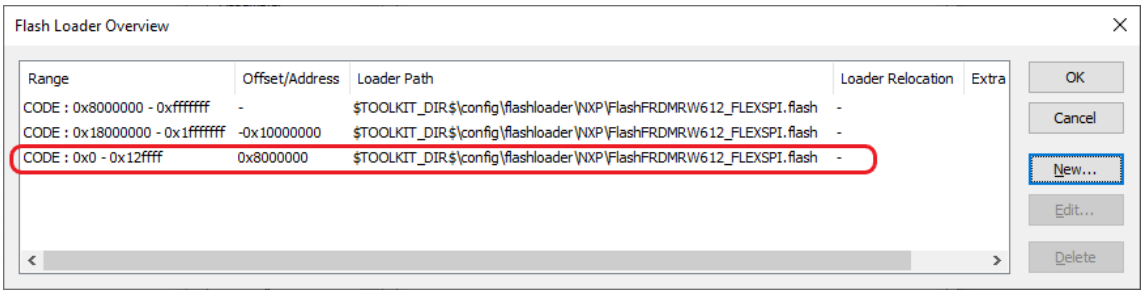
Alternatively, you can use the approach recommended by your IT team to rename the interface to eth0.

Download License Key

To download the license key:

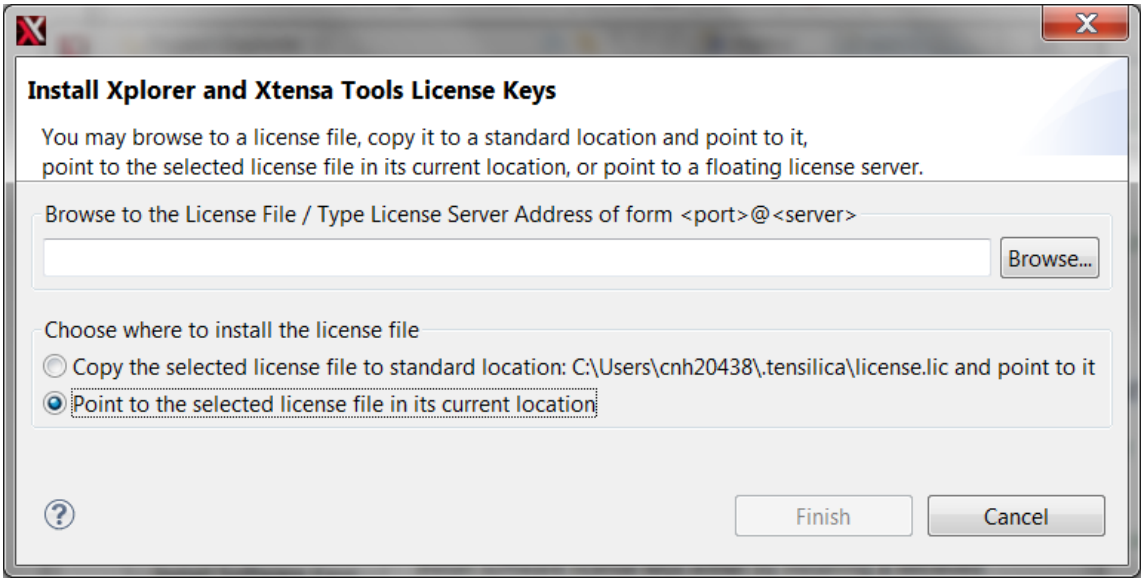
1. Reload or return to the Tensilica URL: https://tensilicatools.com/download/rt600-download-page/.

2. Click the button and get the license key for RT600 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.

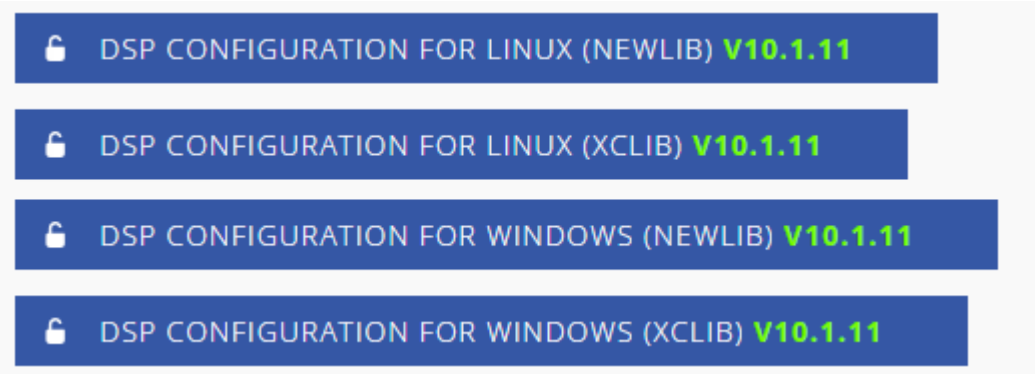
   Note: The generated license file only supports debug/run on the RT600 device target. It does not support software simulation/Xplorer ISS. If you have special requirements to run the software simulations, contact Cadence directly.

Parent topic:*Install Xplorer Toolchains*

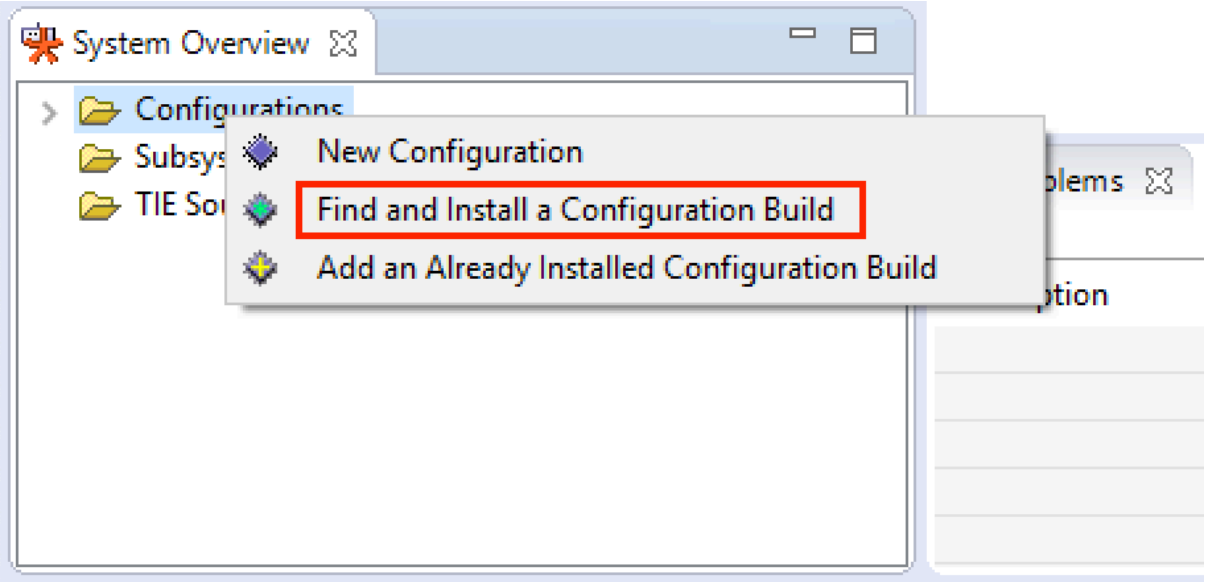
Install RT600 DSP Build Configuration *'Build Configuration'* is a term that describes all parameters and necessary build includes for the Tensilica processor implementation for development.

It is mandatory to install a specific build configuration before starting development on RT600.

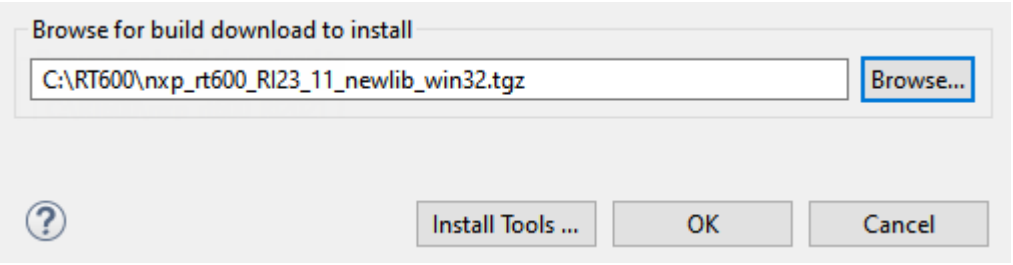
The build configuration is provided by NXP as a binary file and is imported into the Xplorer IDE. The binary file for the OS is available at the Tensilica URL: https://tensilicatools.com/download/rt600-download-page/.



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.



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 RT600. 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.

Note: It is mandatory to replace the 'usbser' section with the 9-digits number J-Link serial number on the back of the J-Link hardware in use.

```
<configuration>
  <controller id='Controller0' module='jlink' usbser='600100000' type='swd' speed='1000000' locking='1'/>
  <driver id='XtensaDriver0' dap='1' xdm-id='12' 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 if Arm RealView ICE (RVI) and DSTREAM debug probes is used.

```
<configuration>
  <controller id='Controller0' module='rvi' />
  <driver id='XtensaDriver0' debug='' inst-verify='mem' module='xtensa' step-intr='mask,stepover,setps'/>
  <driver id='TraxDriver0'   module='trax' />
  <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' xdm-id='12' />
  <device id='Trax0'   component='Component0' driver='TraxDriver0' xdm-id='12' />
  <application id='GDBStub' module='gdbstub' port='20000' >
    <target device='Xtensa0' />
  </application>
  <application id='TraxApp' module='traxapp' port='11444'>
    <target device='Trax0' />
  </application>
</configuration>
```

Congratulations! All Xplorer toolchains are installed.

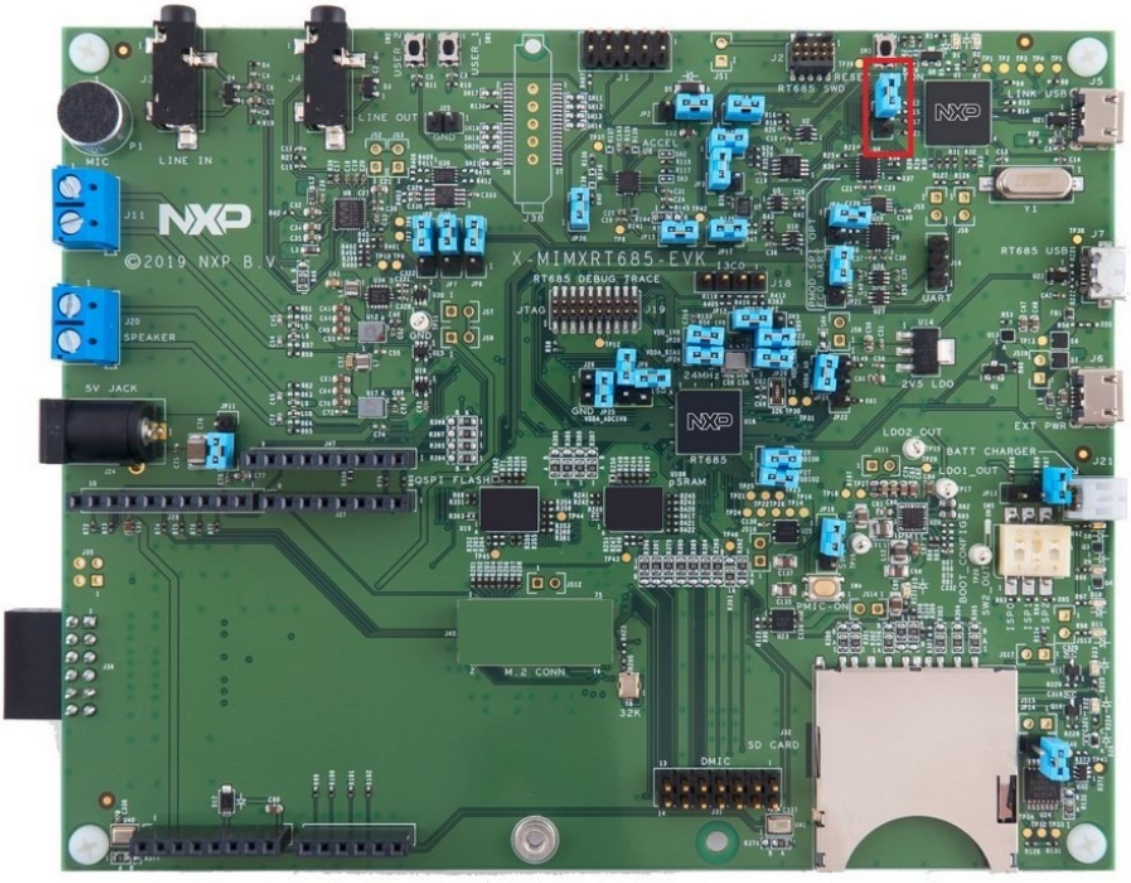
For more details on Xtensa software tools, build configurations, or xt-ocd daemon, see the full set of documents in Xplorer menu **Help > PDF Documentation**.

Program LPC-Link2 with SEGGER J-Link In addition to standalone probes, the onboard LPC-Link2 debug probe is used to debug HiFi4 DSP over SWD ports. The RT600 EVK has an LPC4300 MCU (top right corner on EVK) and by default has been pre-programmed as CMSIS-DAP probe. CMSIS-DAP is not compatible with HiFi4. Therefore, a few extra steps are required to flash it with J-Link firmware and support both Arm CM33 core and HiFi4 DSP core.

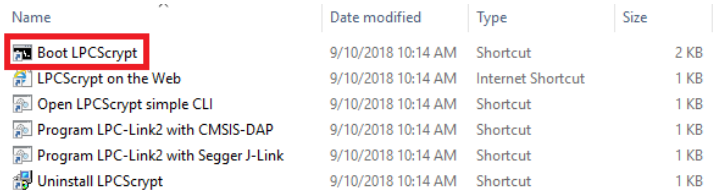
The steps are:

1. Install LPCScrypt, a command-line tool for programming onboard LPC-Link2 debug probe with the latest CMSIS-DAP and J-Link firmware. LPCScrypt is available for download from:https://www.nxp.com/design/microcontrollers-developer-resources/lpc-microcontroller-utilities/lpcscrypt-v2.1.0:LPCSCRYPT.
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 LPC-Link2 debug circuit firmware, unplug the USB cable on J5 and then connect to the DFULink jumper. In the MIMXRT685-EVK, JP1 is the LPCXpresso DFU jumper as shown in Figure 1.
4. Connect JP1 using the jumper.



5. Reconnect the board to the host computer over the debug link USB connector J5.
6. Launch LPCScrypt by double-clicking the *Boot LPCScrypt* file in the LPCScrypt install location: C:\ProgramData\Microsoft\Windows\Start Menu\Programs\LPCScrypt.

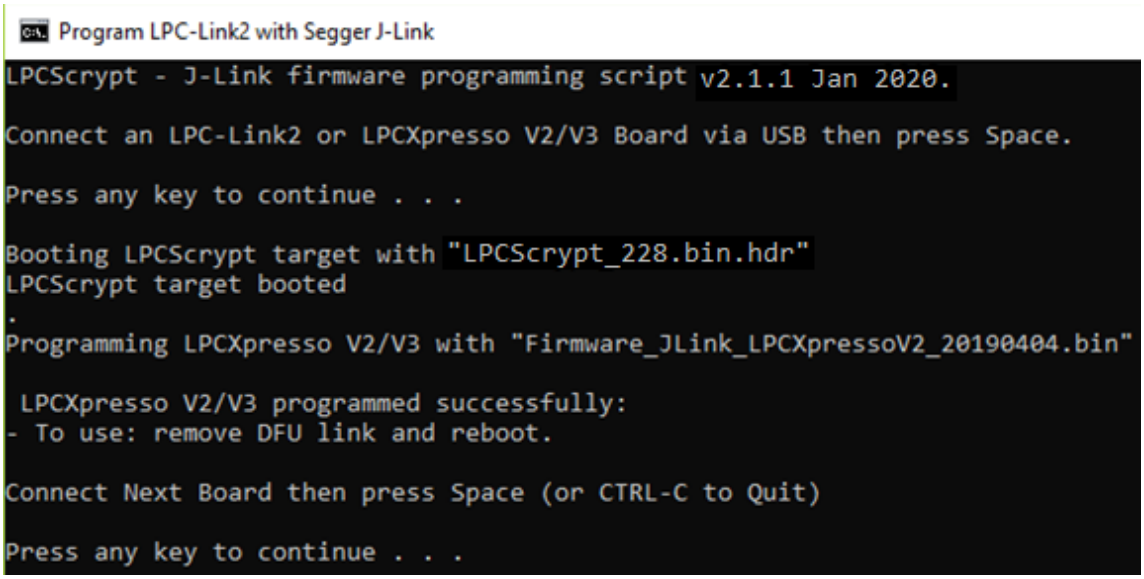


7. In that command shell, run the program_JLINK script to install the JLink debug firmware.

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

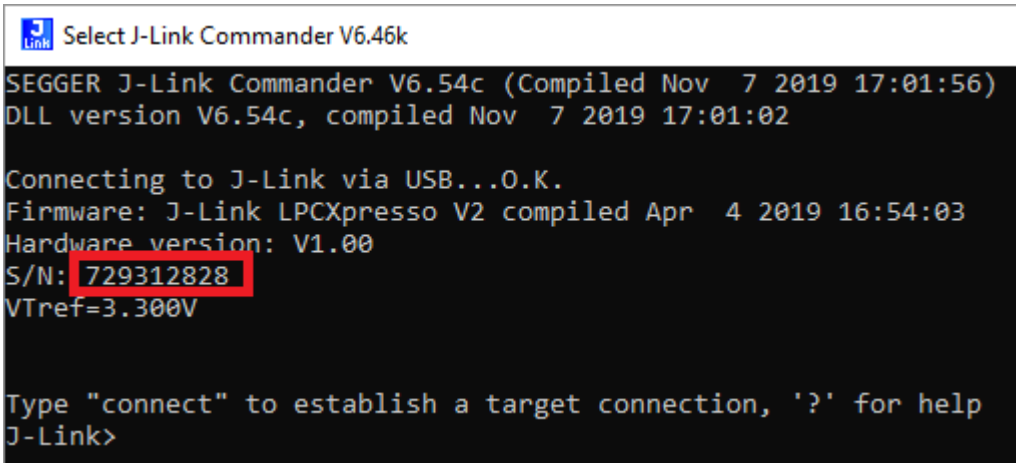
Note: File paths in this document use the Windows directory separator, on Linux or Mac OSX. The file paths must be replaced with '/.' For Windows users, shortcuts to the scripts are available from the LPCScrypt entry on the Start menu.

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

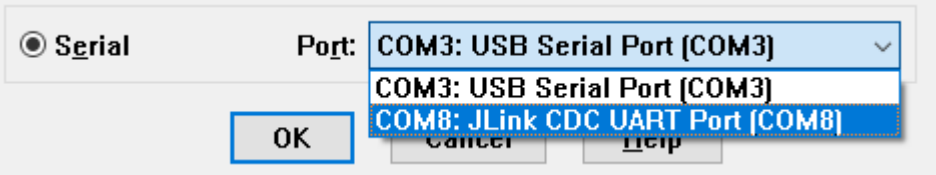


9. Open/ Disconnect JP1 and power cycle the board. The onboard LPC-Link2 is ready to be used as SEGGER J-Link probe.

   Every EVK/ LPC-Link2 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.



10. Another benefit is that LPC-Link2 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 LPC-Link2. If you have any questions, or have difficulties to program the probe, see https://www.nxp.com/design/microcontrollers-developer-resources/lpc-microcontroller-utilities/lpcscrypt-v2.1.0:LPCSCRYPT.

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_rt600_RI23_11_newlib_linux_redist.tgz -C ~/xtensa/builds
# Install the Xtensa development toolchain:
cd ~/xtensa
./builds/RI-2023.11-linux/nxp_rt600_RI23_11_newlib/install \
--xtensa-tools./tools/RI-2023.11-linux/XtensaTools \
--registry ./tools/RI-2023.11-linux/XtensaTools/configg
```

Install MCUXpresso SDK

This section list the steps to:

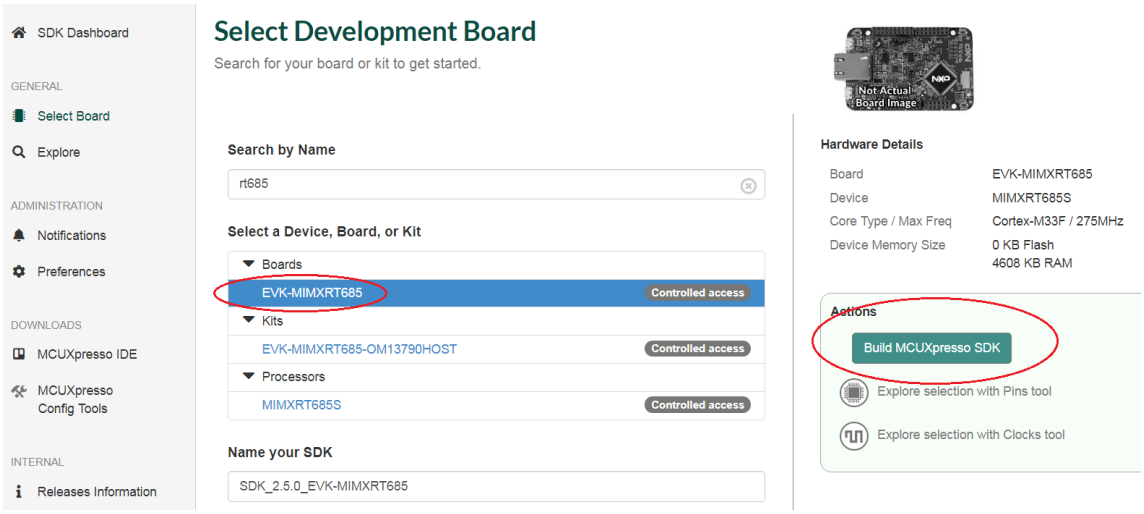
- Download MCUXpresso SDK for RT600
- Enable MCUXpresso SDK DSP
- Initialize DSP Core
- Link DSP Profiles

Download MCUXpresso SDK for RT600 DSP enablement for RT600, including drivers, middleware libraries, and demo applications are included with the latest RT600 SDK available for download from https://mcuxpresso.nxp.com. If you are accessing the site for the first time, make sure to register.

Once logged in, perform the following steps to use the SDK builder.

1. Click the link **Select Board /Processor** on the left panel.
2. In the **Search by Name** field, enter the name of the board. For example, *RT685*.

3. From the search results, select *EVK-MIMXRT685*.
4. Click the **Build MCUXpresso SDK** button on the right panel.



5. In the Build SDK for EVK-MIMXRT685 page, select the required environment settings.
6. Click the **Download SDK** button at the bottom of the page.
7. Save the archive in a local directory.

Enable MCUXpresso SDK DSP The following DSP-specific enablements are available inside the MCUXpresso SDK release package for RT600.

-

<SDK_ROOT>/devices/MIMXRT685S/

Unified device and peripheral driver source code that can be compiled for both Arm and DSP cores.

Note: Only a limited subset of peripheral drivers and components is supported on the DSP.

- ```
<SDK_ROOT>/boards/evkmimxrt685/dsp_examples/
```

DSP example applications

- 

<SDK_ROOT>/middleware/multicore/rpmsg_lite/

Unified RPMsg-Lite multicore communication library, with porting layers for Arm and DSP cores

- ```
<SDK_ROOT>/middleware/dsp/audio_framework/
```

Xtensa Audio Framework \(XAF\) for DSP core

-

<SDK_ROOT>/middleware/dsp/audio_framework/libxa_af_hostless/

```
  Source code and documentation for the core XAF framework

- ```
<SDK_ROOT>/middleware/dsp/audio_framework/testxa_af_hostless/
```

- Utilities and tests for developing applications with the XAF framework

  ```
  <SDK_ROOT>/middleware/dsp/audio_framework/testxa_af_hostless/test/plugins
  ```

- XAF components and codec binaries
-

<SDK_ROOT>/middleware/dsp/naturedsp/hifi4/

```
  NatureDSP Math Library for HiFi4 DSP
```

Initialize DSP Core To minimize the power consumption, the DSP core is not powered when RT600 boots up. To initialize, run, or debug DSP applications, you must execute some code on the Arm core.

The DSP management interface library provided in the SDK is available at the location: <SDK_ROOT>/devices/MIMXRT685S/drivers/fsl_dsp.c.

```
/* Initialize DSP core. */
void DSP_Init(void);
/* Deinit DSP core. */
void DSP_Deinit(void);
/* Copy DSP image to destination address. */
void DSP_CopyImage(dsp_copy_image_t *dspCopyImage);
/* Start DSP core. */
void DSP_Start(void);
/* Stop DSP core. */
void DSP_Stop(void);
```

The SDK includes a helper function used by the DSP example applications at: <SDK_ROOT>/boards/evkmimxrt685/dsp_examples/dsp_support.c.

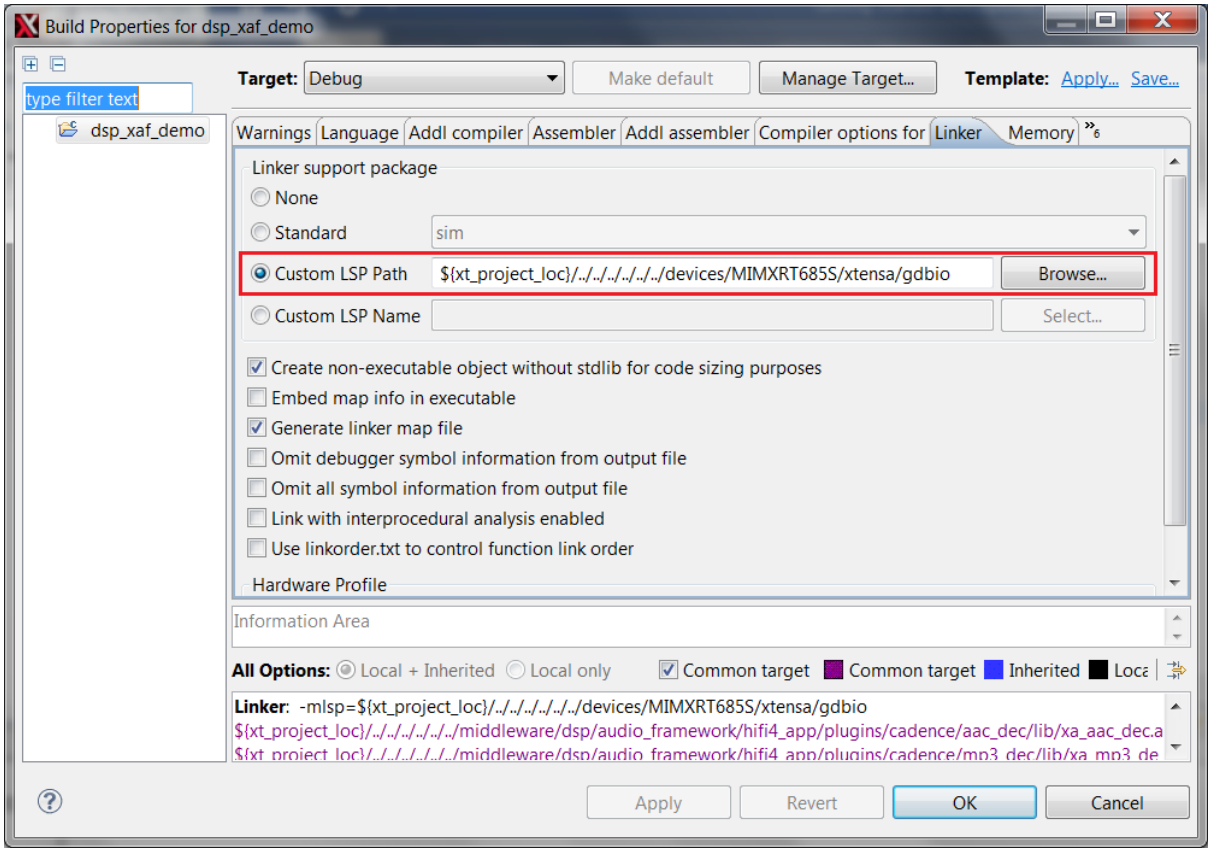
```
/* Prepare DSP core for code execution:
  - Setup PMIC for DSP
  - Initialize DSP clock and core
  - (Optional) Copy DSP binary image into RAM
  - Start DSP core
*/
void BOARD_DSP_Init(void);
```

After executing this function during your Arm application startup, the DSP is initialized and ready to run. From here, code is loaded and debugged on the DSP with Xplorer IDE and tools.

Link DSP Profiles The Xtensa Software Tools use the linker support packages (LSPs) to link a HiFi4 DSP application for the RT600. An LSP includes both a system memory map and a collection of libraries to include into the final binary. The LSPs are provided in the MCUXpresso SDK and are available at: <SDK_ROOT>/devices/MIMXRT685S/xtensa/.

DSP sample applications are configured to link against one of these custom LSPs. By default, *Debug* targets are linked against the gdbio LSP intended for use with an attached debugger and captures I/O requests (printf) through gdb. The *Release* target links against the min-rt LSP, which includes minimal runtime support.

To change the LSP used by a project target in the Xplorer IDE, use the **Linker** tab in the project **Build Properties** dialog box.



The MCUXpresso SDK ships with other standard LSPs for RT600. For more information on LSPs and creating a custom memory map using Xtensa software tools, see the Cadence Linker Support Packages (LSPs) Reference Manual.

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'
- Run and Debug DSP Audio Framework
- Launch DSP Application from Arm Core

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 in the manner described in Initialize 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.

- Build and execute the 'Hello World' Arm demo located in: <SDK_ROOT>/boards/evkmimxrt685/dsp_examples/hello_world_usart/cm33/.

Preparing an Arm core development environment is outside the scope of this document. For information on how to use the SDK for Arm core development, see the *Getting Started with MCUXpresso SDK for MIMXRT600.pdf* document at <SDK_ROOT>/docs/.

Note: IAR Embedded Workbench may require a patch to enable compatibility with RT600. For details on the patch, contact NXP directly.

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

Start Xtensa Debugger Daemon To debug DSP applications on RT600, ensure that the xt-ocd daemon is running. This application runs a gdb server that the Xtensa core debugger connects to.

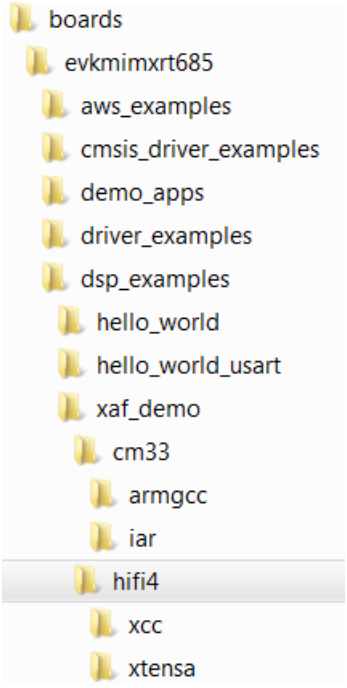
Go to the command-line window and change directory (cd) to xt-ocd daemon installation path. The default path on Windows is C:\Program Files (x86)\Tensilica\Xtensa OCD Daemon 14.08.

Execute the daemon with the custom topology.

```
**xt-ocd.exe -c topology.xml**
XOCD <VERSION_INFO>
(c) 1999-2021 Cadence Design Systems Inc. All rights reserved.
[Debug Log 2021-06-17 08:12:29]
Loading module "gdbstub" <VERSION_INFO>
Loading module "jlink" <VERSION_INFO>
Using JLINK lib <VERSION_INFO>
Jlink USB Serial Number: XXXXXXXXX
Connected to Jlink Device:
  Name:'J-Link LPCXpresso V2 compiled Apr  4 2019 16:54:03'
  S/N:XXXXXXXX
  Firmware: J-Link LPCXpresso V2 compiled Apr  4 2019 16:54:03
  Requested/Set TCK: 2000kHz/65534kHz
Jlink: Select SWD
SWD-DP with ID 0x6BA02477
Loading module "jtag" <VERSION_INFO>
Loading module "xtensa" <VERSION_INFO>
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. For details on xt-ocd runtime options and configuration, see Chapter 7 of the Xtensa Debug Guide, available in **Help** > **PDF Documentation**.

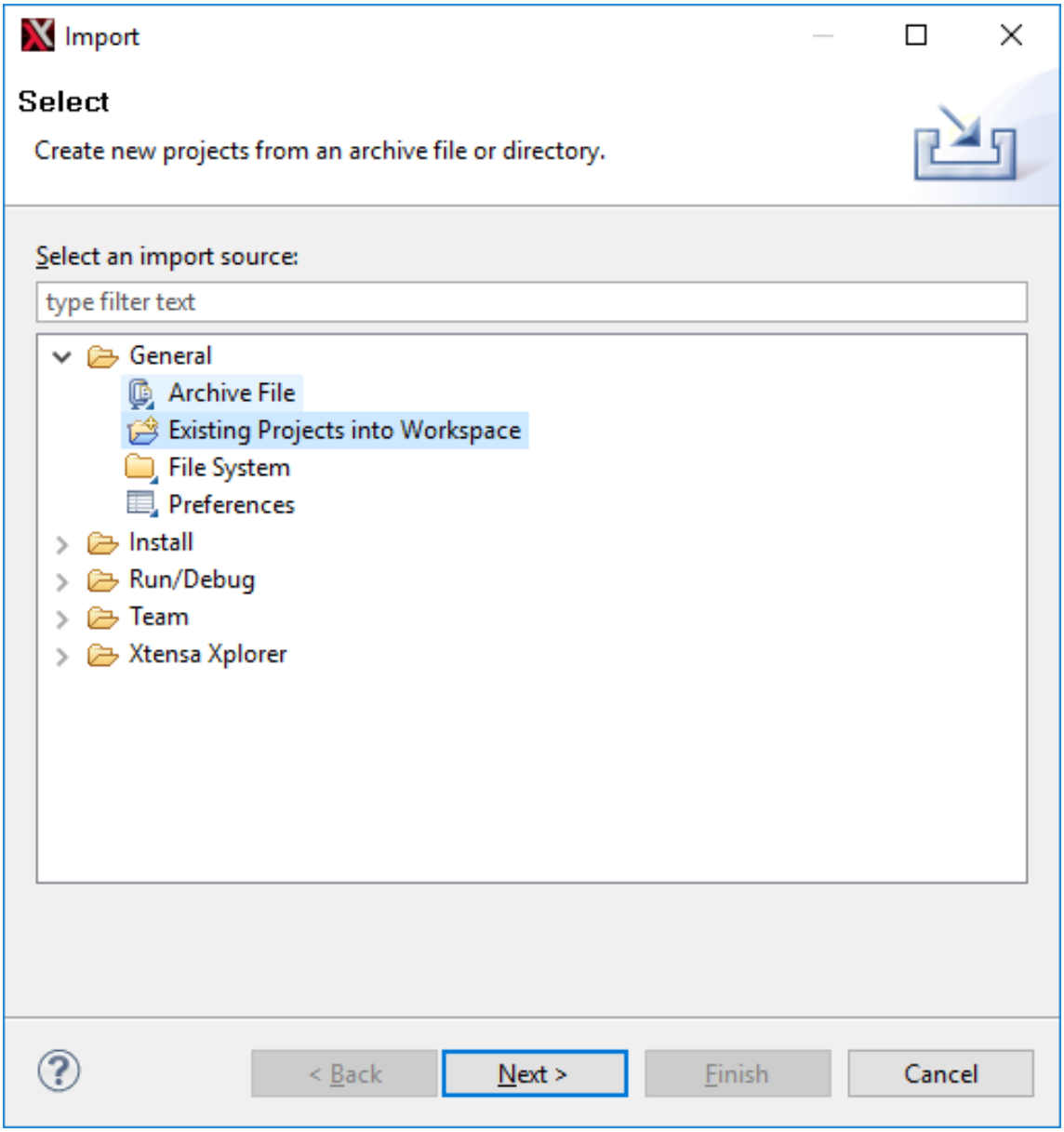
Prepare DSP Core for 'Hello World' The RT600 SDK provides a collection of DSP example applications located in boards/evkmimxrt685/dsp_examples/. Each DSP example has two source directories, one for the Arm Cortex-M33 core ('cm33') and one for the DSP HiFi4 core ('dsp').



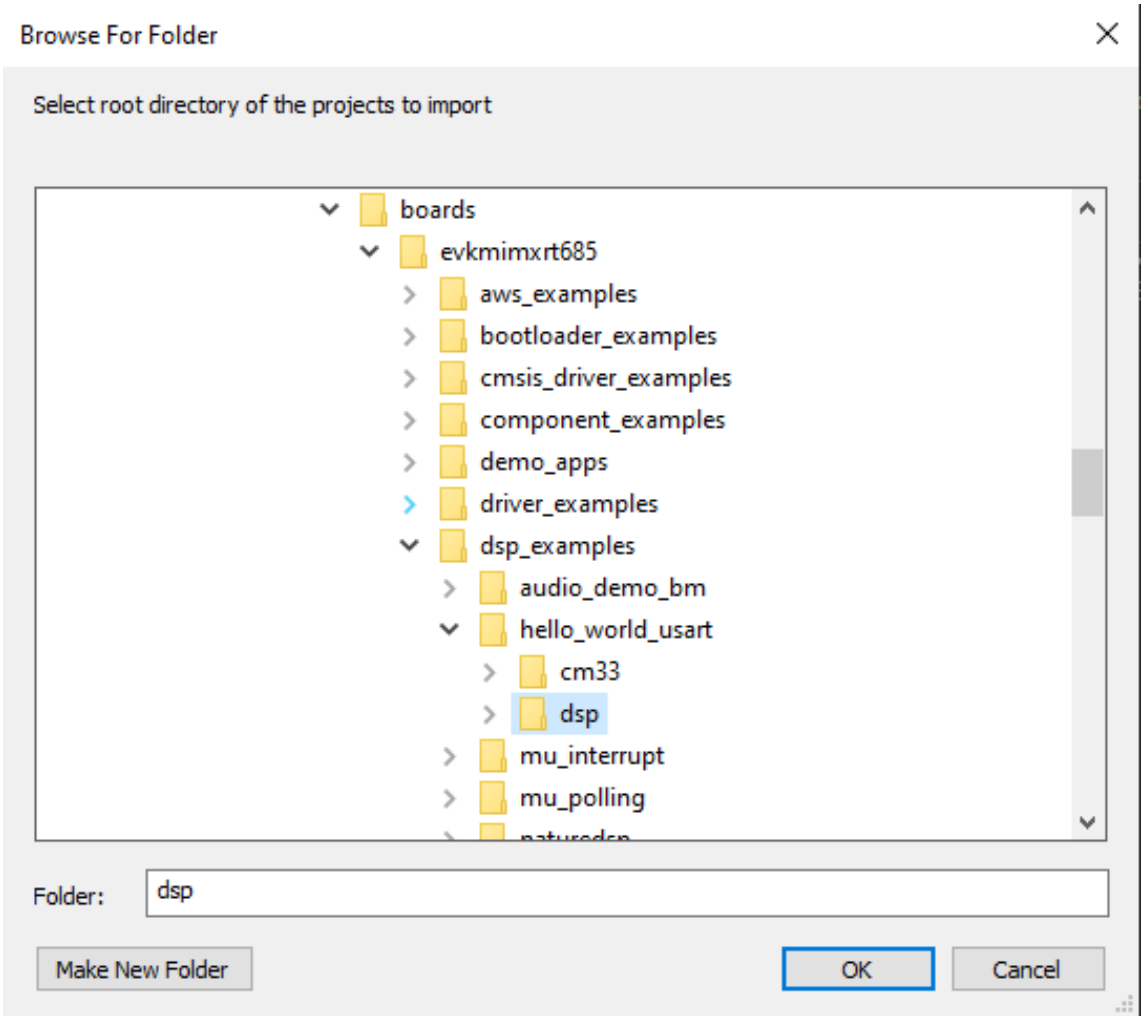
The projects for different supported toolchains are build in the above directories. For the DSP example above, the 'xcc' project builds on the command line and the 'xtensa' directory is an Xplorer IDE project.

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

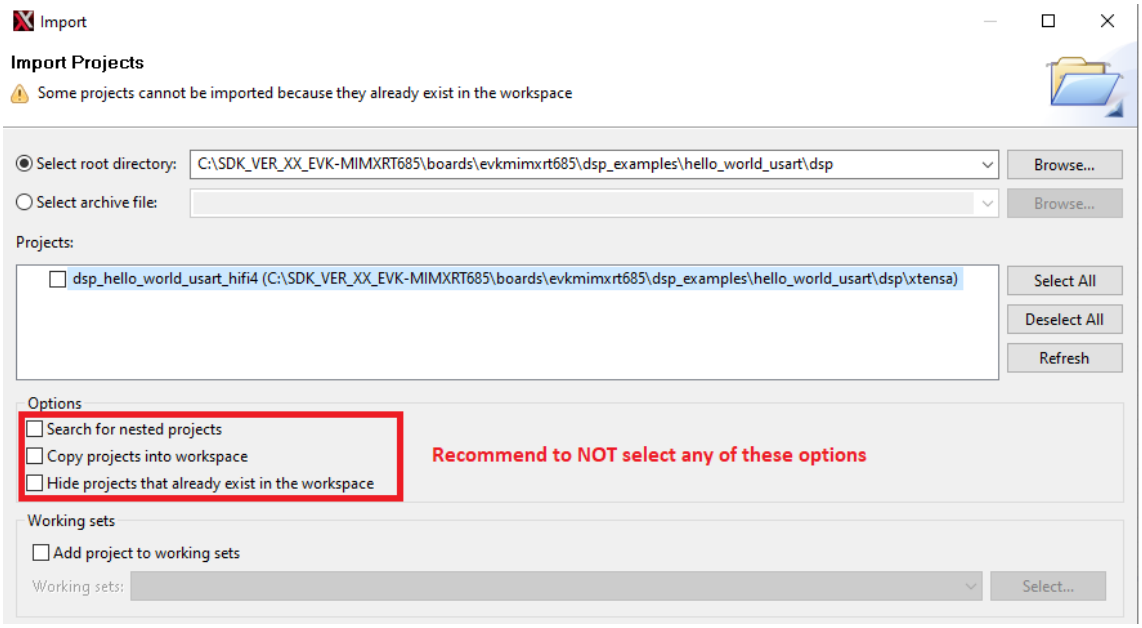
1. Select **File** > **Import** > **General** > **Existing Projects into Workspace**.



2. Select the SDK directory <SDK_ROOT>\boards\evkmimxrt685\dsp_examples\hello_world_usart\dsp\xtensa as root directory.

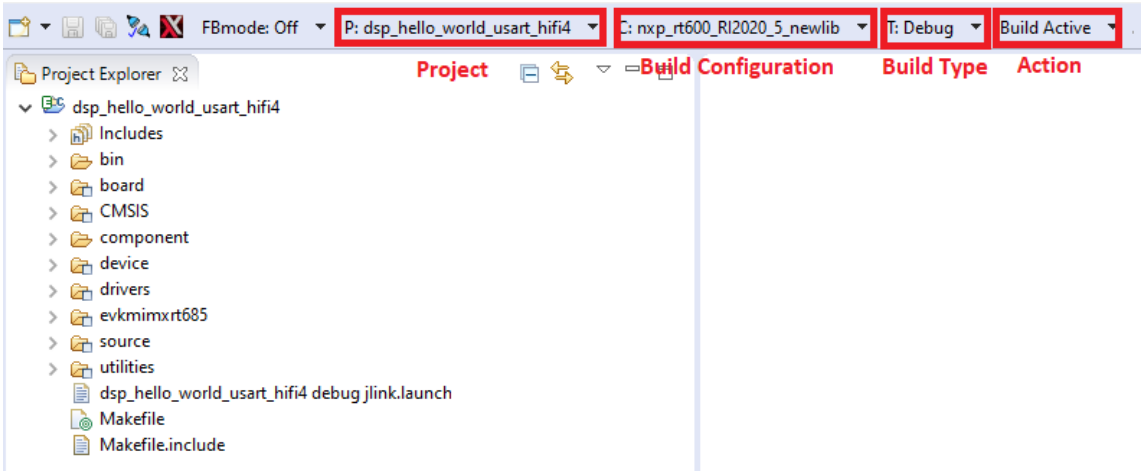


3. Leave all the other check boxes blank.

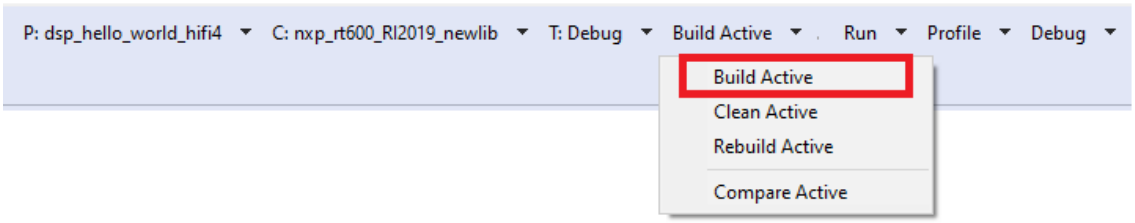


Once imported, the 'dsp_hello_world_usart_hifi4' appears in the Project Explorer.

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

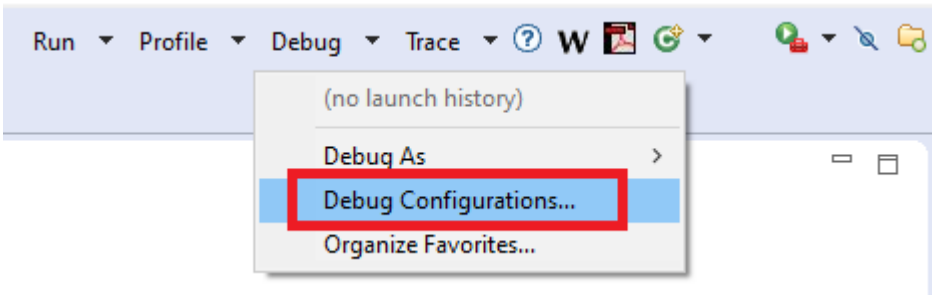


5. To build the project for debug, profile, or trace, use the action buttons on the right side of the menu bar.



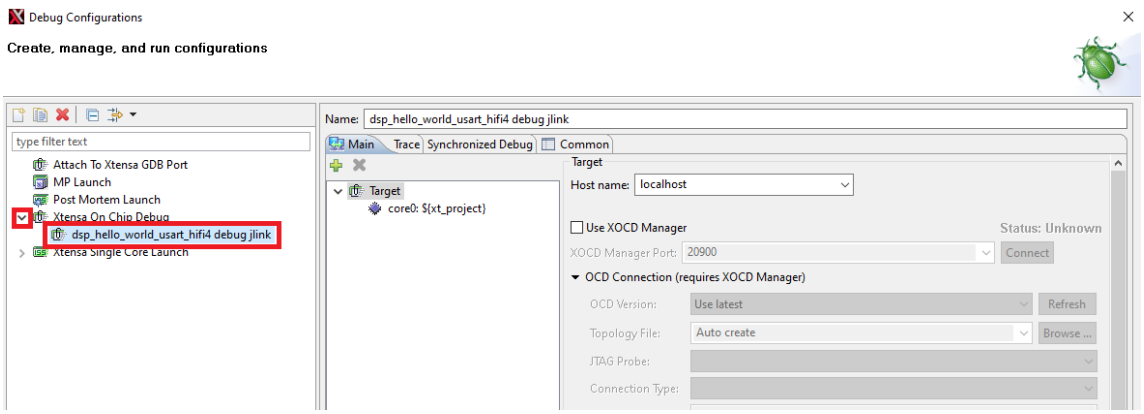
A default debug configuration is provided by the SDK project, which utilizes the on-chip debugger.

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

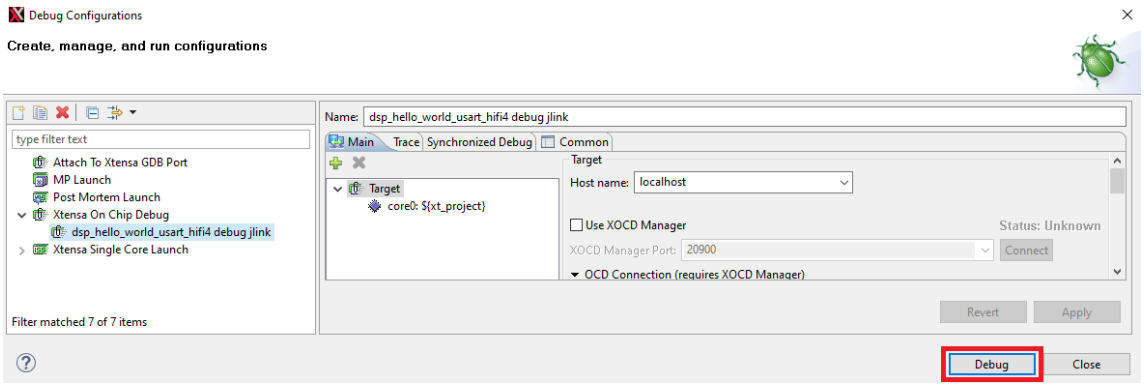


The **Debug Configurations** dialog box appears.

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

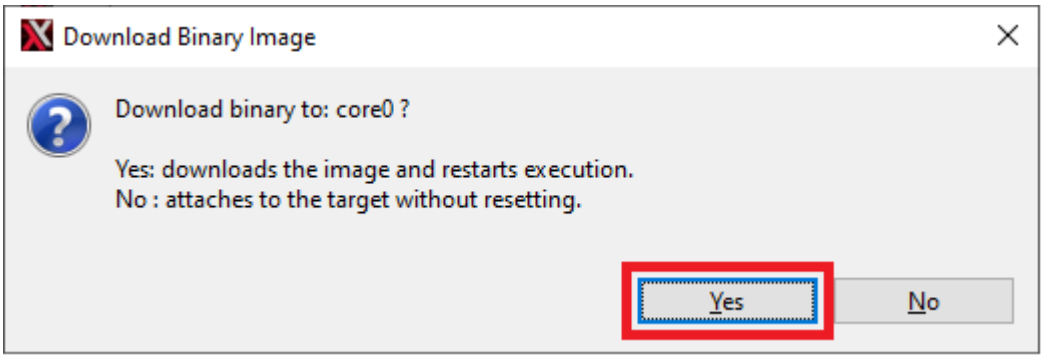


8. Click the **Debug** button. The actual debug on the chip initiates.



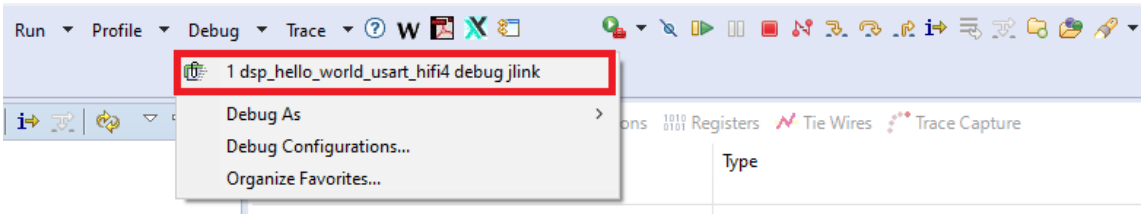
The Xplorer IDE prompts whether binaries should download to the hardware.

9. Select **Yes**.



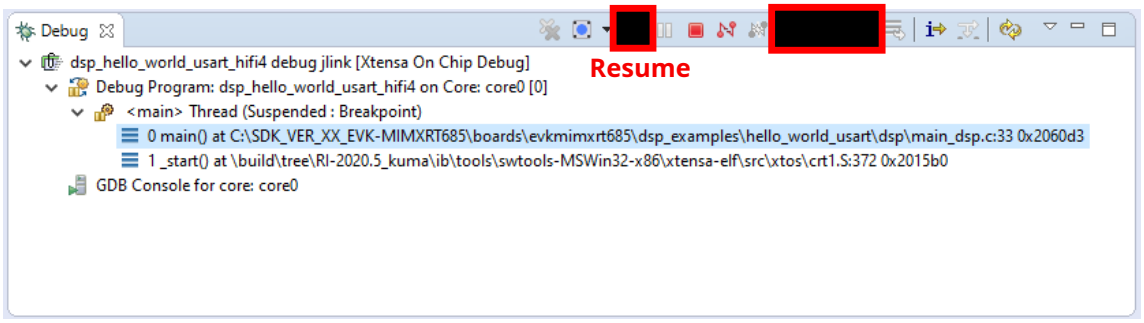
10. The Xplorer IDE transitions to the **Debug** perspective after the binary download.

    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*.

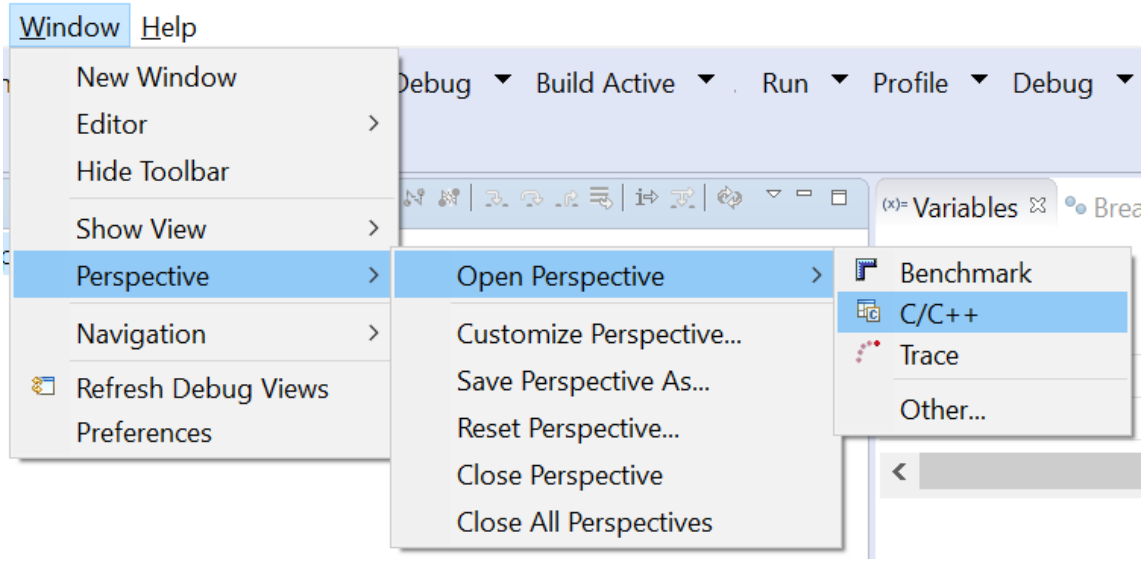
The program stops at the start of main() function. To run the program, click the **Resume** or **Stepping through** icon button as shown in *Figure 10*.



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

```
Hello World running on core nxp_rt600_RI2020_5_newlib
```

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*

Run and Debug DSP Audio Framework The DSP audio framework demo consists of separate applications that run on the Arm core and DSP core. The Arm application runs a command shell and relays the input requests to the DSP application using RPMsg-Lite.

EVK Board Setup for Audio Demo

The DSP audio demo is tested against EVK-MIMXRT685 Rev E and requires UART for serial console.

For the codec to output audio properly, attach one jumper on the board as follows:

JP7-1 <==> JP8-2

Note: DSP operates DMA and audio peripherals same as CM33 side. Jumper settings, I2S configs, or pin mux settings are all same with CM33 side. For information on steps to operate I2S, DMIC and DMA, see the SDK CM33 side driver examples. SDK\boards\evkmimxrt685\driver_examples\i2s;

Note: Jumper settings are different on different version of EVKs or validation boards. The above mentioned setting is for SDK 2.8.0 and Rev E EVK or later only. For old version of boards or SDKs, double check the SDK driver examples for correct settings for the pin mux and jumper settings.

The demo uses the UART for console input and output. Connect the EVK board to a PC via the USB debug interface (J5) and open up a serial interface on your PC using a terminal tool such as PuTTY on Windows or screen on Linux.

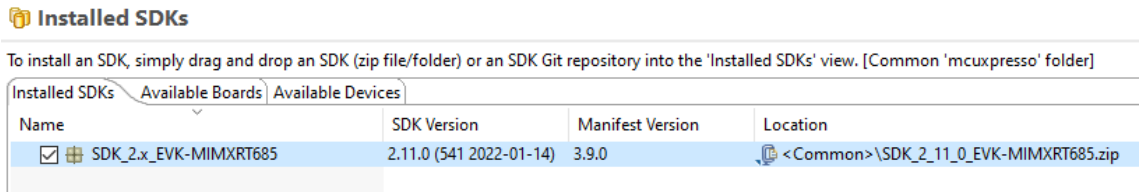
Debug Audio Demo

To debug this DSP application, set up and execute the Arm application in a desired environment. For more information on Arm development environment options, see 'Getting Started with MCUXpresso SDK for EVK-MIMXRT685.pdf.

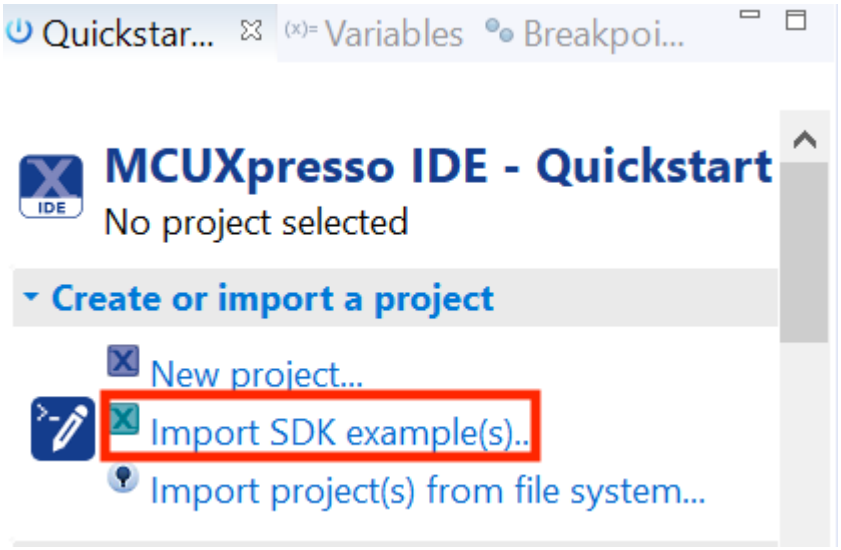
The following example uses the NXP MCUXpresso IDE for the Arm environment.

To debug the audio demo, perform the following steps.

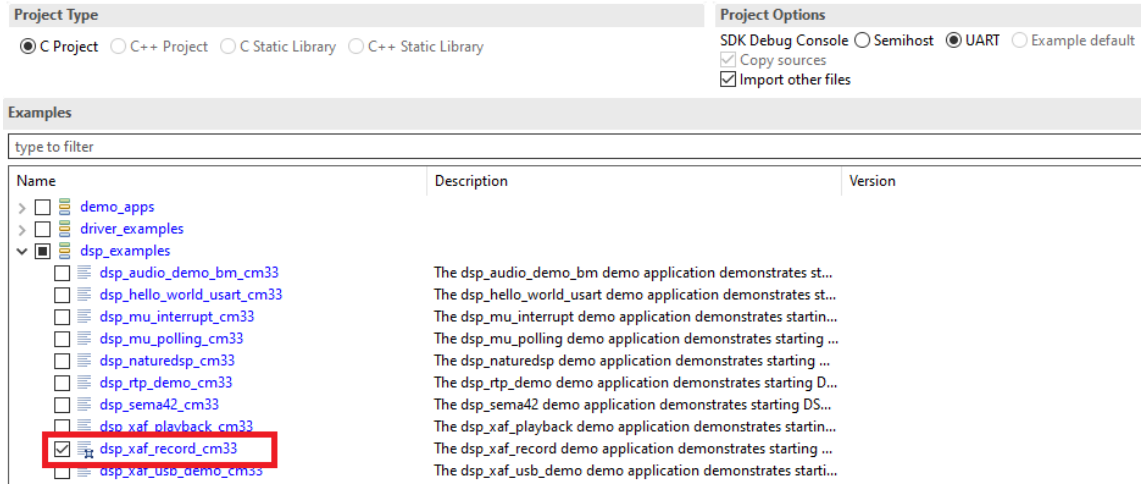
1. Install the MCUXpresso SDK for RT600 into the MCUXpresso IDE using the Installed SDK panel as shown in



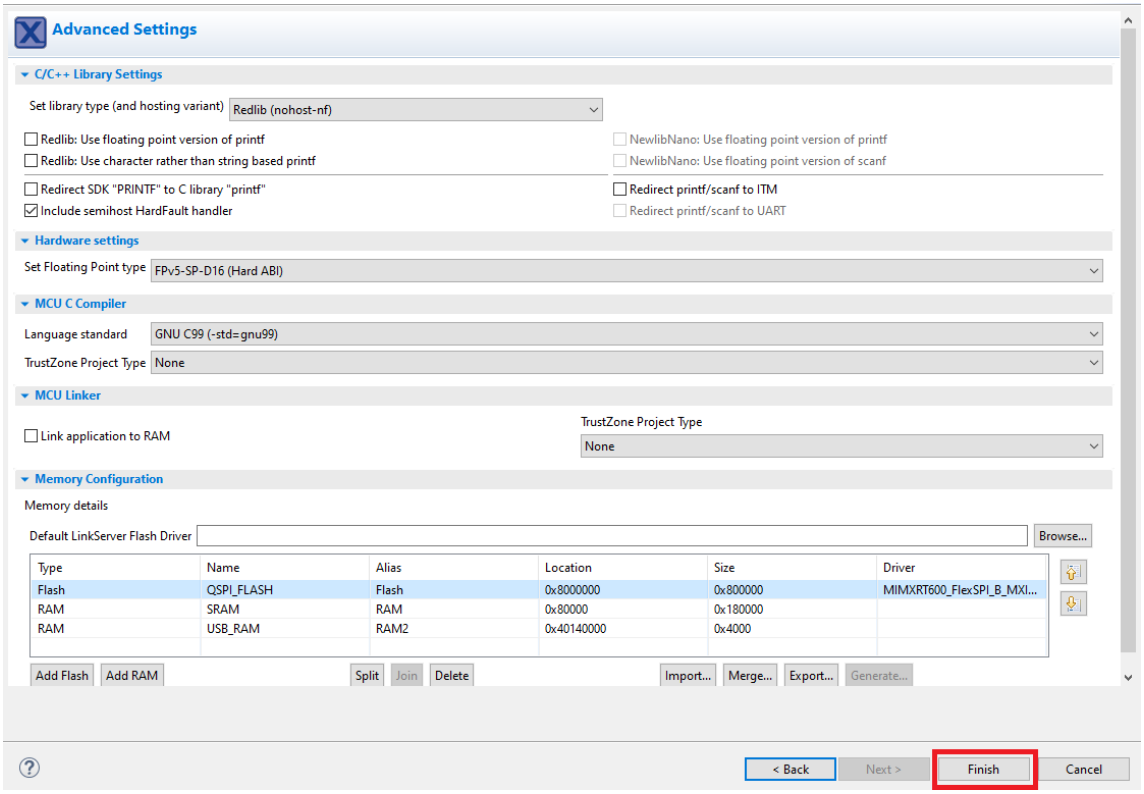
2. To import an example from the installed SDK., use the **Import SDK examples** link in the **Quickstart** panel on the lower left of the screen.



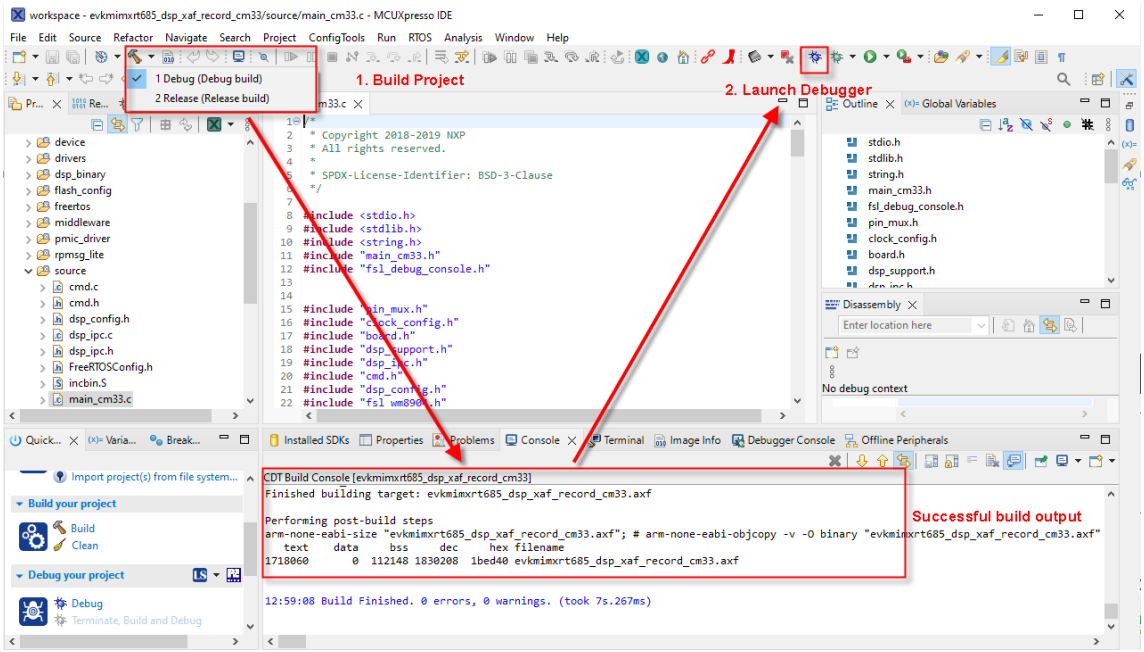
3. Select the '*dsp_xaf_record_cm33*' example for Cortex-M33 core.



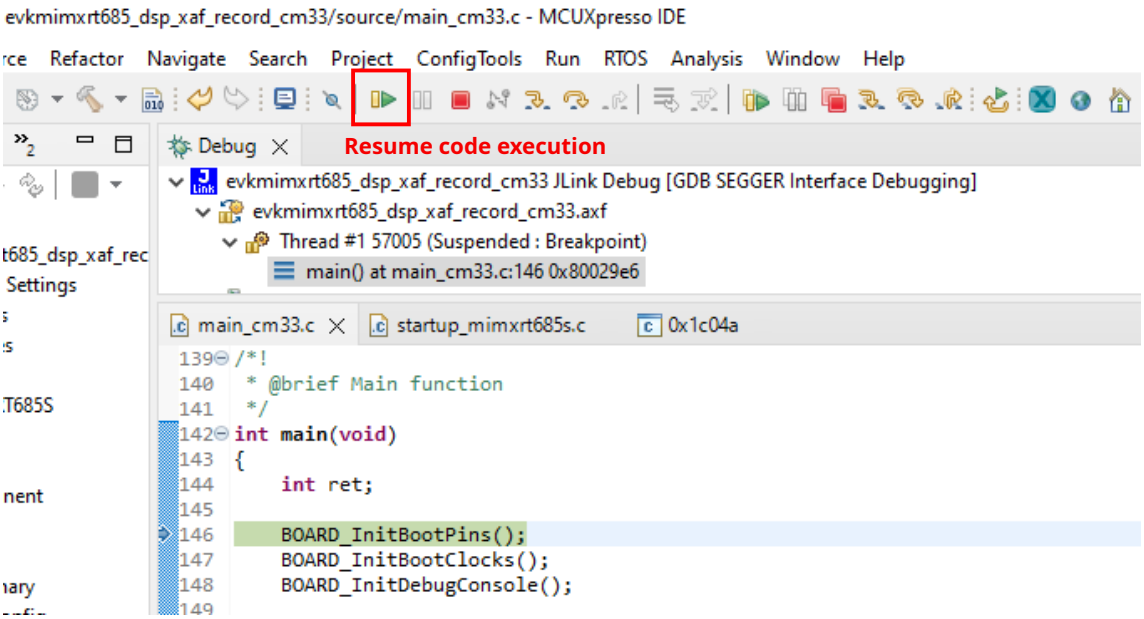
4. Click the **Finish** button.



5. Build the project and launch the debugger on success.



6. Use the debug toolbar to resume the code execution.



7. Observe serial terminal output with shell prompt.

```
******************************
DSP audio framework demo start
******************************
Configure WM8904 codec
DSP image copied to DSP TCM
[APP_DSP_IPC_Task] start
[APP_Shell_Task] start
Copyright 2022 NXP
>>
```

8. Using the Xplorer IDE, load and execute xaf_record using the steps described in Prepare DSP Core for 'Hello World'.

9. After the DSP application runs, use the serial shell to invoke the 'record_dmic [language]' command.For information on the supported languages, check VIT ReleaseNotes.txt. Using the serial shell creates an audio pipeline that captures microphone audio, perform voice recognition (VIT), and playback via the codec speaker line out (J4 on the EVK).

```
>> help
"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 the wake-word and in 3 s say the frame supported command. If selected model contains strings, then the wake-word and the list of commands appears in the console.

Note: this command does not return to the shell

10. The serial terminal shows the IPC communications to set up an audio device..

```
>> record_dmic en

Command not recognized.  Enter 'help' to view a list of available commands.

>> record_dmic en
[CM33 CMD] Setting VIT language to en
[DSP_Main] Number of channels 2, sampling rate 16000, PCM width 16
[CM33 CMD] [APP_DSP_IPC_Task] response from DSP, cmd: 19, error: 0
[DSP Record] Audio Device Ready
[CM33 CMD] DSP DMIC Recording started
[DSP VoiceSeeker] VoiceSeekerLight_GetRequiredHeapMemoryBytes: 51120 bytes
[CM33 CMD] To see VIT functionality say wakeword and command
[DSP VoiceSeeker] VoiceSeekerLight_Create: voiceseeker_status = 0
[DSP VoiceSeeker] VoiceSeekerLight_GetLibVersion: v0.6.0
```

The VIT wake-word and supported commands appear in the serial console.

```
[DSP VIT]   VIT Model Release = 0x40800
[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'
[DSP VIT]    'WATCH MOVIE'
[DSP Record] connected CAPTURER -> GAIN_0
[DSP Record] connected XA_GAIN_0 -> XA_VOICE_SEEKER_0
[DSP Record] connected XA_VOICE_SEEKER_0 -> XA_VIT_PRE_PROC_0
[DSP Record] connected XA_VIT_PRE_PROC_0 -> XA_RENDERER_0
```

The serial console prints the detected wake-word and commands.

```
- WakeWord detected
- Voice Command detected 8 POWER ON
- WakeWord detected
- Voice Command detected 2 NEXT
```

For more information on configuration and using the Audio Framework demo, see the file <SDK_ROOT>\boards\evkmimxrt685\dsp_examples\xaf_record\cm33\readme.txt..

For more details about VIT which supports English and Mandarin models, see <SDK_ROOT>\middleware\vit\HIFI4\Doc\VIT_Integration_Guide.pdf.

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

Launch DSP Application from Arm Core In the previous example, the Arm application and DSP application are independently loaded and debugged. This section list the steps to produce one Arm application binary that includes and starts the DSP application without the use of a debugger or a loader.

The Arm core application for each DSP demo uses a global preprocessor macro to control loading of the DSP binary application:

DSP_IMAGE_COPY_TO_RAM

When this macro is set to '1'/TRUE (default value), it instructs the DSP demo application to do the following:

- Link the DSP application binary images into the Arm binary
- Copy the DSP application images into RAM on program boot
- Initialize the DSP to run from the RAM image

Note: When modifying the Arm project, make sure to supply the global preprocessor macro to both the C compiler and assembler.

To build the DSP application image for the Arm application, select the **Release** target option in the Xplorer IDE. For more information on building with min-rt LSP, see section Link DSP Profiles.

Three DSP binaries are generated and are loaded into different TCM or SRAM address segments:

- <SDK_ROOT>/boards/evkmimxrt685/dsp_examples/xaf_record/dsp/binary/**dsp_data_release.bin**
- <SDK_ROOT>/boards/evkmimxrt685/dsp_examples/xaf_record/dsp/binary/**dsp_text_release.bin**
- <SDK_ROOT>/boards/evkmimxrt685/dsp_examples/xaf_record/dsp/binary/**dsp_ncache_release.bin**

  Note: Depending on the environment, there might be a need to copy the binary images into the Arm application workspace.

Run and Debug from Command-Line Environment / LINUX

RT600 SDK has been configured to be as flexible to support multiple toolchains, including the Arm GCC and XCC command-line environments. The principles and essentials are still the same as with the IDE. The command-line environment settings are nearly identical between WIN32 and LINUX, with few different path settings.

Build and Debug Arm Application The Arm application requires the GNU Arm Embedded Toolchain and CMake version 3.x for command-line compile and linking. For more information on installation and configuration of required build tools for command-line development, see the *Getting Started with MCUXpresso SDK for EVK_MIMXRT685.pdf* in the *<SDK_ROOT>/docs/* directory.

To build and debug:

1. Launch a command prompt / terminal and change directory to the xaf_record application.

```
user@linux:~/SDK/boards/evkmimxrt685/dsp_examples/xaf_record/cm33/armgcc$ ls -1
build_all.bat
build_all.sh
build_debug.bat
build_debug.sh
build_flash_debug.bat
build_flash_debug.sh
build_flash_release.bat
build_flash_release.sh
build_release.bat
build_release.sh
clean.bat
clean.sh
CMakeLists.txt
```

2. Use .bat files to build the configuration on Windows, and .sh files on Linux/UNIX.

```
user@linux:~/SDK/boards/evkmimxrt685/dsp_examples/xaf_record/cm33/armgcc$ ./build_debug.sh
...
[100%] Linking C executable debug/dsp_xaf_record_cm33.elf
[100%] Built target dsp_xaf_record_cm33.elf
```

3. Launch the GDB server.

```
user@linux:/opt/JLink$ ./JLinkGDBServerCLExe -device MIMXRT685S_M33 -if SWD
SEGGER J-Link GDB Server V6.46j Command Line Version
...
Listening on TCP/IP port 2331
Connecting to target...Connected to target
Waiting for GDB connection...
```

4. Connect with GDB to the device and load Arm application.

```
user@jlinux:~/SDK/boards/evkmimxrt685/dsp_examples/xaf_record/cm33/armgcc$ arm-none-eabi-
↪gdb debug/dsp_xaf_record_cm33.elf
...
Reading symbols from debug/dsp_xaf_record_cm33.elf...
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x1301ec7a in ?? ()
(gdb) mon reset
Resetting target
(gdb) load
Loading section .flash_config, size 0x200 lma 0x7f400
Loading section .interrupts, size 0x130 lma 0x80000
Loading section .text, size 0xe330 lma 0x80130
Loading section CodeQuickAccess, size 0x52c lma 0x8e460
Loading section .ARM, size 0x8 lma 0x8e98c
Loading section .init_array, size 0x4 lma 0x8e994
Loading section .fini_array, size 0x4 lma 0x8e998
Loading section .data, size 0x104 lma 0x8e99c
Start address 0x801e4, load size 60576
Transfer rate: 272 KB/sec, 5506 bytes/write.
(gdb) b main
Breakpoint 1 at 0x808f2: file /SDK/boards/src/dsp_examples/xaf_record/cm33/main_cm33.c, line 161.
(gdb) c
Continuing.
Breakpoint 1, main ()
    at /SDK/boards/src/dsp_examples/xaf_record/cm33/main_cm33.c:161
161     BOARD_InitHardware();
```

Build and Debug DSP Application The Xtensa command-line toolchain is installed as part of the Xplorer IDE. The tools can be optionally installed on a new Windows or Linux system without the IDE using the redistributable compressed file found at: <XTENSA_ROOT>/XtDevTools/downloads/RI-2023.11/tools/. For more information, see Install Xtensa On Chip Debugger Daemon.

In order to use the command-line tools, some environment variables must be set up for use with the cmake build scripts:

```
# Add tools binaries to PATH.  Assume ~/xtensa/ is install root - please adjust accordingly.
export PATH=$PATH:~/xtensa/XtDevTools/install/tools/RI-2023.11-linux/XtensaTools/bin
# (Optional) Use environment variable to control license file
# NOTE: ~/.flexlmrc will override this selection.  Please delete that file before proceeding.
export LM_LICENSE_FILE=~/RT600.lic
# Setup env vars needed for compile and linking
export XCC_DIR=~/xtensa/XtDevTools/install/tools/RI-2023.11-linux/XtensaTools
export XTENSA_SYSTEM=~/xtensa/XtDevTools/install/builds/RI-2023.11-linux/
nxp_rt600_RI23_11_newlib/configexport XTENSA_CORE=nxp_rt600_RI23_11_newlibb
```

Note: On Windows, you can use the 'setx' command instead of the 'export' command to set the environment variables.

1. Use the batch/shell script to build out the DSP application from the command line, in the 'xcc' directory:

```
user@linux:~/SDK/boards/evkmimxrt685/dsp_examples/xaf_record/dsp/xcc$ ./build_debug.sh
...
[100%] Built target dsp_xaf_record_hifi4.elf
```

   Note: Some warnings during the linking process (floating-point ABI) may appear. However, the warnings may be ignored.

2. Launch xt-ocd debugging server (replace topology.xml with your custom version – see section 1.5 of this document):

```
user@linux:/opt/Tensilica/xocd-14.11$ ./xt-ocd.exe -c topology.xml
```

   Note: If the Arm core fails to initialize the DSP, the xt-ocd daemon may fail to start. Therefore, the Arm core must initialize the DSP first.

3. Connect with Xtensa GDB to the device and execute the DSP application:

```
user@linux:/SDK/boards/evkmimxrt685/dsp_examples/xaf_record/dsp/xcc$ xt-gdb
debug/dsp_xaf_record_hifi4.elf
GNU gdb (GDB) 7.11.1 <Xtensa Tools VERSION NUMBER>
...
Reading symbols from debug/dsp_xaf_record_hifi4.elf...done.
(xt-gdb)
(xt-gdb) target remote localhost:20000
Remote debugging using localhost:20000
_DoubleExceptionVector ()
    at /home/xpgcust/tree/RI-2023.11/ib/tools/swtools-x86_64-linux/xtensa-elf/src/xos/src/xos_
↪vectors.S:216
216 /home/xpgcust/tree/RI-2023.11/ib/tools/swtools-x86_64-linux/xtensa-elf/src/xos/src/xos_vectors.
↪S: No such file or directory.
(xt-gdb) reset
_ResetVector ()
    at /home/xpgcust/tree/RI-2023.11/ib/tools/swtools-x86_64-linux/xtensa-elf/src/xtos/xea2/reset-
↪vector-xea2.S:71
71  /home/xpgcust/tree/RI-2023.11/ib/tools/swtools-x86_64-linux/xtensa-elf/src/xtos/xea2/reset-
↪vector-xea2.S: No such file or directory.
(xt-gdb) load
Loading section .rtos.rodata, size 0x80 lma 0x200000
Loading section .rodata, size 0x17d50 lma 0x200080
Loading section .text, size 0x633f0 lma 0x217dd0
Loading section .rtos.percpu.data, size 0x4 lma 0x27b1c0
Loading section .data, size 0x110c lma 0x27b1d0
Loading section NonCacheable, size 0x2960 lma 0x20040000
Loading section .Level3InterruptVector.literal, size 0x4 lma 0x24000000
Loading section .DebugExceptionVector.literal, size 0x4 lma 0x24000004
Loading section .NMIExceptionVector.literal, size 0x4 lma 0x24000008
Loading section .ResetVector.text, size 0x13c lma 0x24020000
Loading section .WindowVectors.text, size 0x16c lma 0x24020400
Loading section .Level2InterruptVector.text, size 0x1c lma 0x2402057c
Loading section .Level3InterruptVector.text, size 0xc lma 0x2402059c
Loading section .DebugExceptionVector.text, size 0xc lma 0x240205bc
Loading section .NMIExceptionVector.text, size 0xc lma 0x240205dc
Loading section .KernelExceptionVector.text, size 0xc lma 0x240205fc
Loading section .UserExceptionVector.text, size 0x18 lma 0x2402061c
Loading section .DoubleExceptionVector.text, size 0x8 lma 0x2402063c
Start address 0x24020000, load size 520016
Transfer rate: 8 KB/sec, 10612 bytes/write.
```

(continues on next page)

(continued from previous page)

```
(xt-gdb) b main
Breakpoint 1 at 0x21ab5b: file /SDK/boards/src/dsp_examples/xaf_record/hifi4/xaf_main_hifi4.c,␣
↪line 366.
(xt-gdb) c
Continuing.
Breakpoint 1, main ()
    at /SDK/boards/src/dsp_examples/xaf_record/hifi4/xaf_main_hifi4.c:366
366     xos_start_main("main", 7, 0);
(xt-gdb) c
Continuing.
```

Note: You can use the gdb command 'set substitute-path' to map the missing symbols from the toolchain libraries. For example:

```
set substitute-path /home/xpgcust/tree/RI-2023.11/ib/tools/swtools-x86_64-linux ~/xtensa/tools/RI-
↪2023.11-linux/XtensaTools
```

For more information on xt-gdb, see the Cadence GNU Debugger User's Guide and the Cadence Xtensa Debug Guide. The documents are available at:

- ~/xtensa/XtDevTools/downloads/RI-2023.11/docs/gnu_gdb_ug.pdf
- ~/xtensa/XtDevTools/downloads/RI-2023.11/docs/xtensa_debug_guide.pdf

HiFi4 System Programming

This section provides more examples, tips, and some best practices about HiFi4 programming on RT600 EVKs. It focuses more on RT6xx and SDK. For general HiFi programming, see the Xtensa IDE documents. You can find them in the directory **Xtensa Xplorer IDE menu Help > PDF Documentation**.

The following are some frequently used references:

- Xtensa Instruction Set Architecture (ISA) Reference Manual: Architecture/ high-level overview.
- HiFi 4 DSP User's Guide: Most useful reference manual for DSP programmer. It has all details about HiFi4 instructions and intrinsics, as well as some algorithm optimization techniques.
- Xtensa XOS Reference Manual: XOS is the default and native embedded kernel for Xtensa HiFi cores. SDK examples use XOS as well.
- Xtensa System Software Reference Manual: XTOS, also known as the basic runtime and handlers, has been depreciated and moved toward XOS. However, the Xtensa Processor Hardware Abstraction Layer/ HAL is still very useful in many perspectives. SDK examples use HAL functions.
- Xtensa Linker Support Packages (LSPs) Reference Manual and GNU Linker User's Guide: Both documents are useful for understanding the HiFi program linker and the memory map.

Specifically for RT6xx, the user manual and the data sheet are the most important references. All product specifications are found in the user guide and data sheet documents.

HiFi4 Boot Loader and Memory Map This section provides an overview of:

- HiFi4 Boot Loader
- Linker and Memory Map
- Cache and Data Exchange Memory Partitions

- Boot or Run from Flash

HiFi4 Boot Loader

For better power efficiency, by default the HiFi4 DSP is powered off when RT6xx powers up. To boot up, Cortex M33 acts as master core to configure DSP local memories, clocks, load DSP images, and so on. SDK has wrapped up this part as HiFi4 Boot Loader or the DSP driver. The essentials are located in: <SDK path>\devices\MIMXRT685S\drivers\fsl_dsp.h & fsl_dsp.c.

For each DSP example, it also provides two more implementation files for detailed configurations: <SDK path>\boards\evkmimxrt685\dsp_examples\<any dsp example>\cm33\dsp_support.h & dsp_support.c

Below is the DSP boot procedure with more elaborations.

To run DSP at full power, for example, 600 MHz, SoC Vddcore is set to 1.1 V. If full power is not required, for example, to run DSP at the half speed at 300 MHz, then Vddcore only requires 0.8 V. RT6xx EVK integrates NXP PCA9420 PMIC for power management and by default Vddcore is set to 1.0 V.

Therefore, PMIC is initialized for better power management. For details on Vddcore and DSP frequency operating conditions, see any dsp example\cm33\pmic_support.c BOARD_SetPmicVoltageForFreq(), or see the data sheet, section 13.1, General Operating Conditions.

```
/* Initialize PMIC PCA9420 */
BOARD_InitPmic();
/* Configure PMIC Vddcore value according to main/dsp clock. */
BOARD_SetPmicVoltageForFreq(CLOCK_GetMainClkFreq(), CLK_600MHZ);
```

It is possible to clock DSP from various clock sources. The DSP PLL for full power and can also run at FFRO low-frequency clocks to save the power.

```
/* Enable DSP PLL clock 594MHz. */
CLOCK_InitSysPfd(kCLOCK_Pfd1, 16);
/*Let DSP run on DSP PLL clock with divider 1 (594Mhz). */
CLOCK_AttachClk(kDSP_PLL_to_DSP_MAIN_CLK);
CLOCK_SetClkDiv(kCLOCK_DivDspCpuClk, 1);
```

As Cortex M33 and SRAM are clocked at lower speed/ max frequency at 300 MHz, set the DSP AHB bus clock divider as 2.

```
CLOCK_SetClkDiv(kCLOCK_DivDspRamClk, 2);
```

If DSP clock is running at 300 MHz or lower, it is more efficient to use divider as 1. For divider as 1, note that an extra register SYSCTL0_PACKERENABLE is set. For more details, see the user manual section 4.5.2.18 DSP Main Ram Clock Divider and section 4.5.5.3 Packer Enable.

```
/* This is a quick register setting example for secure mode */
/* SYSCTL0->PACKERENABLE = 0x4 */
```

Power up TCM/ DSP local memories and cache, supply clock, and reset peripherals.

```
/* Initializing DSP core */
DSP_Init();
```

For SDK DSP examples, split the DSP images into three parts.

One is for vectors and critical sections sitting on TCM/ DSP local memories. The second one is for normal code and data sections sitting on SRAM, and the final is for non-cached DSP initialized data in SRAM.

Here, Cortex M33 load those binaries to its destination. When the DSP program is debugged, it is possible to load DSP binaries from the Xtensa Xplorer IDE, as described in Prepare DSP Core

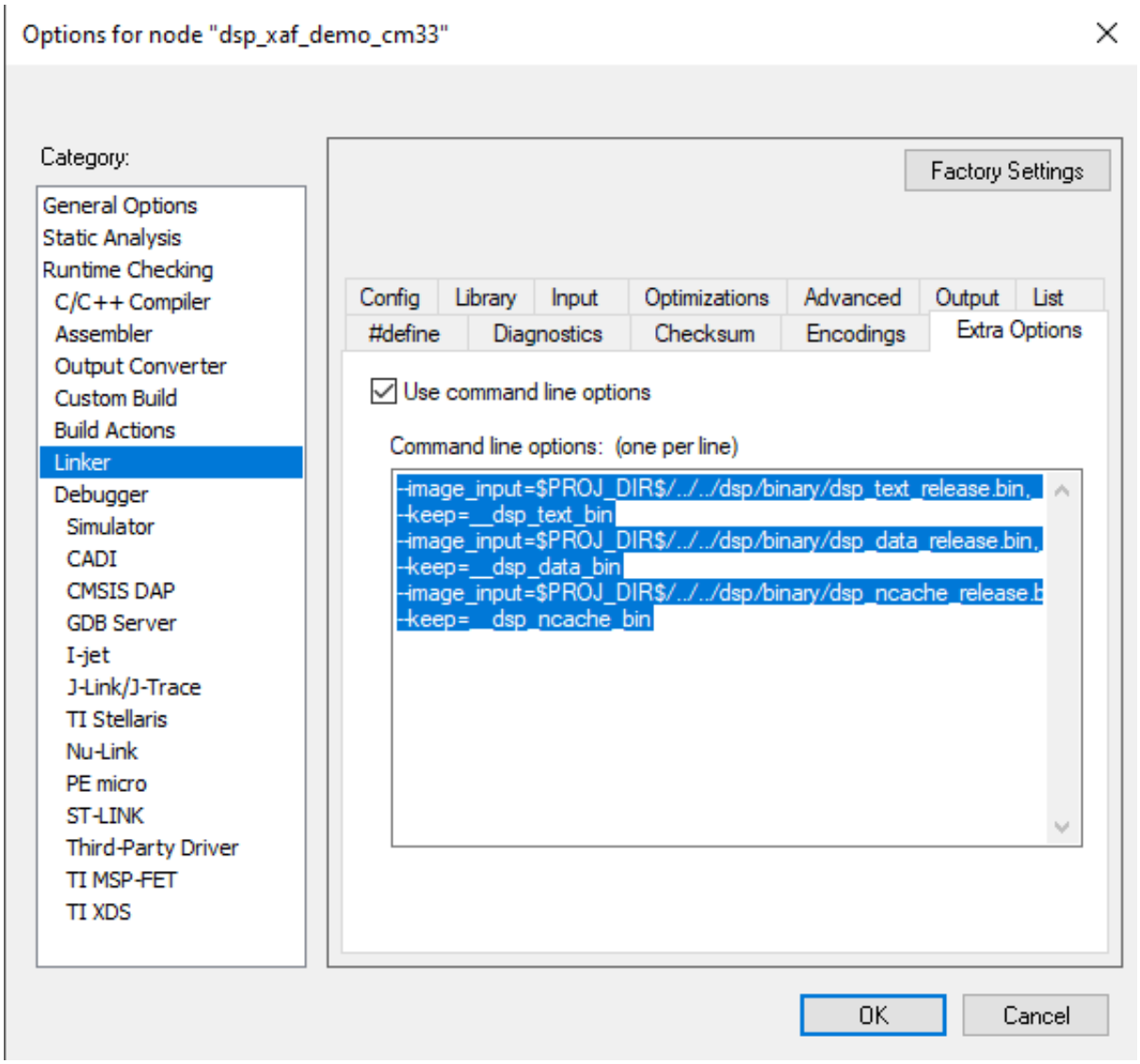
for 'Hello World'. To load binaries, remove the DSP_IMAGE_COPY_TO_RAM compilation flag or set it to 0. By default, the compilation flag is set to 1 and always load the DSP images.

```
#if DSP_IMAGE_COPY_TO_RAM
    /* Copy application from RAM to DSP_TCM */
    DSP_CopyImage(&tcm_image);
    /* Copy application from RAM to DSP_RAM */
    DSP_CopyImage(&sram_image);
    /* Copy application from RAM to DSP_Uncached RAM */
    DSP_CopyImage(&ncache_image);
#endif
```

The DSP stall register SYSCTL0_DSPSTALL controls the HiFi4 operation. Start the DSP and run it.

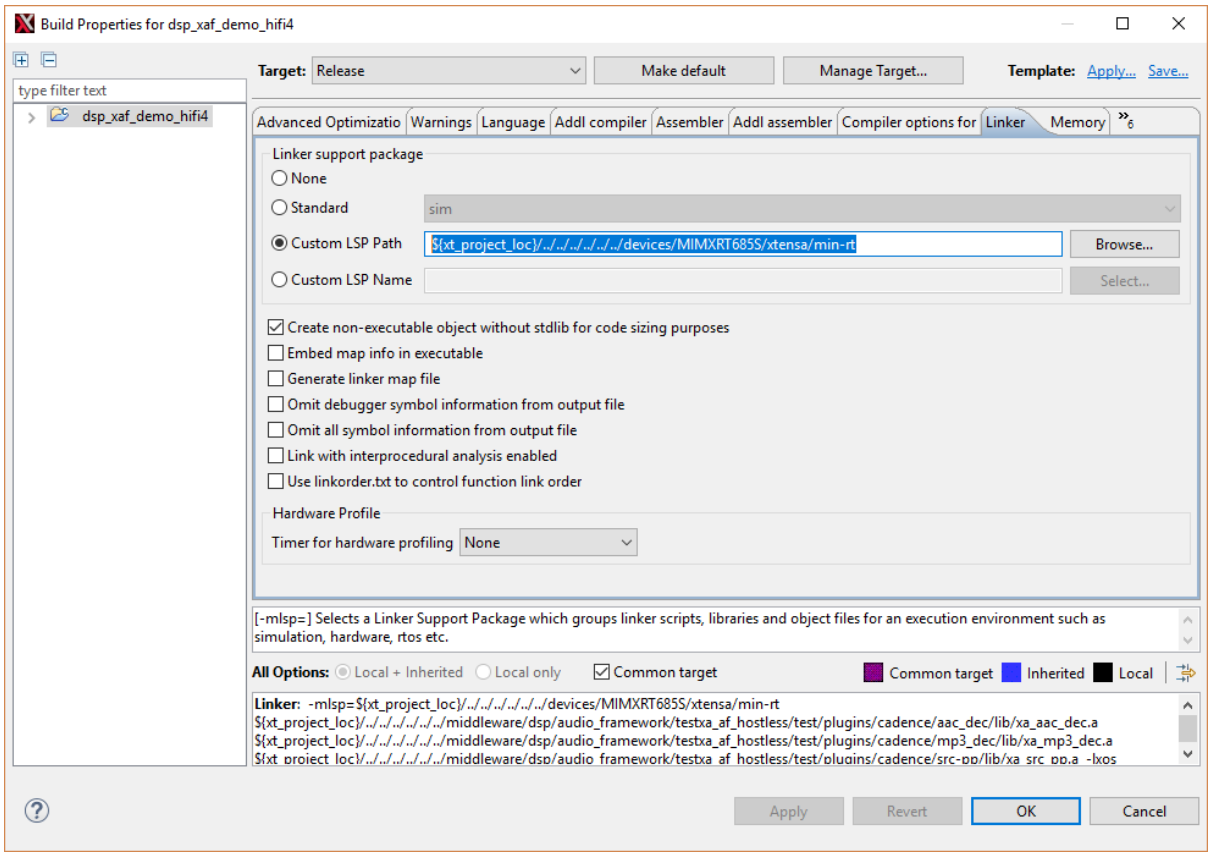
```
/* Run DSP core */
DSP_Start();
```

The post build scripts create the DSP images. For details, see Makefile.include in any DSP example. To reduce the image size and make image copy more efficient, they are split into SRAM part; TCM part, and uncached SRAM part. DSP images are set and linked into Cortex M33 side. For IAR, the linker is set in **Project Options** > **Linker** > **Extra Options**. For ArmGCC or Linux environment, set in any DSP example\cm33\incbin_gcc.S.



Linker and Memory Map

When importing the SDK DSP examples, by default the mode is set as Release. The default mode means that the images must be built in Release mode with 'min-rt' Linker Support Package/ LSP. To double-check, open **SDK DSP examples** > **Build Properties** > **Linker**.



SDK provides three different LSPs.

- **'min-rt' for Release mode** - 'min-rt' eliminates all unnecessary debug info and reduces image size.
- **'gdbio' for Debug mode** - gdbio' support standard 'printf'/ log output back to Xtensa Xplorer debug console, as well as other debug utilities, perfect for debug purpose but not appropriate for official deployment nor loading directly from Cortex M33 side.
- **'sim' for simulations** - 'sim' only works for software simulation and does not fit on device debugging.

The memory map is identical for different LSPs. It sits with linker scripts in SDK path\devices\MIMXRT685S\xtensa\'LSP name'\ldscripts\elf32xtensa.x. It specifies how HiFi4 DSP organizes image sections on the memory. For example:

- 0x0020 0000 ~ 0x0048 0000, size 2.5 M bytes, for code and data.
- Stack and Heap are at the top of the segment and count top down from 0x0048 0000 to lower.
- 0x2400 0000 ~ 0x2400 FFFF, size 64 K bytes, for Data TCM. By default, it is empty.
- 0x2402 0000 ~ 0x2402 FFFF, size 64 K bytes, for instruction TCM. By default it only contains essential vectors and left around 62 K for applications.
- 0x2004 0000 ~ 0x2007 FFFF, size 256 K bytes. This is non-cached area for Cortex M33 and HiFi4 DSP data exchange.

Note that both Cortex M33 and HiFi4 DSP have access to all SRAM partitions. It means that a unified memory map is necessary at system level and both cores must not affect each other's memory map. For SDK examples, see that HiFi4 memory map starts from 0x0020 0000 and Cortex M33 side sits under this address. Using IAR environment, as an example, its memory map sits in SDK path\boards\evkmimxrt685\dsp_examples\ any example\cm33\iar\MIMXRT685Sxxxx_ram.icf.

- 0x0008 0000 ~ 0x0017 FFFF, for interrupt vectors and code.
- 0x0018 0000 ~ 0x001F FFFF, for data.

The memory map is flexible and can be adjusted as per the requirement of the application. Note that modifying the core's memory map might affect another. Changes to both the cores must be made accordingly. For example, when allocating more SRAM partitions to DSP, you must reduce the memory taken at Cortex M33 side. Otherwise, Cortex M33 might not work properly. Also, when loading DSP image directly from Cortex M33 side, the image still sits in Cortex M33 data section before booting up. Therefore, it raises the bar for application data section requirements. You may consider running the application from FLASH.

Cache and Data Exchange Memory Partitions

You may have noticed that HiFi4 DSP has a small non-cached area that starts from 0x20040000. The non-cached are used for data exchange between two cores. As both M33 and HiFi DSP have shared access to all SRAM partitions, shared memory access is the most effective way to exchange data between two cores. The given physical addresses are read/ written by both cores at same address, no memory mapping or address converting is required. For example, if a piece of data array is passed from Cortex M33 to HiFi4 DSP, only the start pointer and the size of the array is passed, and conversely. It is convenient for system programming and simplifies the inter-core communications.

Consider cache here. Cortex M33 has no cache, the entire SRAM is considered as its local memory. Therefore, any memory write is flushed immediately. HiFi4 has 32 K instruction cache and 64 K data cache, and both cache are enabled by default. Therefore, the memory write is not flushed immediately. To make a tradeoff between performance and IPC convenience, set the non-cached area for data exchange memory partitions.

The above memory map has specified the non-cached region, and in DSP code, and HAL functions are called to disable the cache. For details, see the audio framework example in SDK path\boards\evkmimxrt685\dsp_examples\xaf_demo\dsp\xaf_main_dsp.c. For more details about HAL cache function, see the Xtensa System Software Reference Manual, section 3.11 Cache

```
/* Disable DSP cache for RPMsg-Lite shared memory. */
xthal_set_region_attribute((void *)RPMSG_LITE_SHMEM_BASE, RPMSG_LITE_SHMEM_SIZE,␣
↪XCHAL_CA_BYPASS, 0);
/* Disable DSP cache for noncacheable sections. */
xthal_set_region_attribute((uint32_t *)&NonCacheable_start,
                           (uint32_t)&NonCacheable_end - (uint32_t)&NonCacheable_start, XCHAL_CA_
↪BYPASS, 0);
```

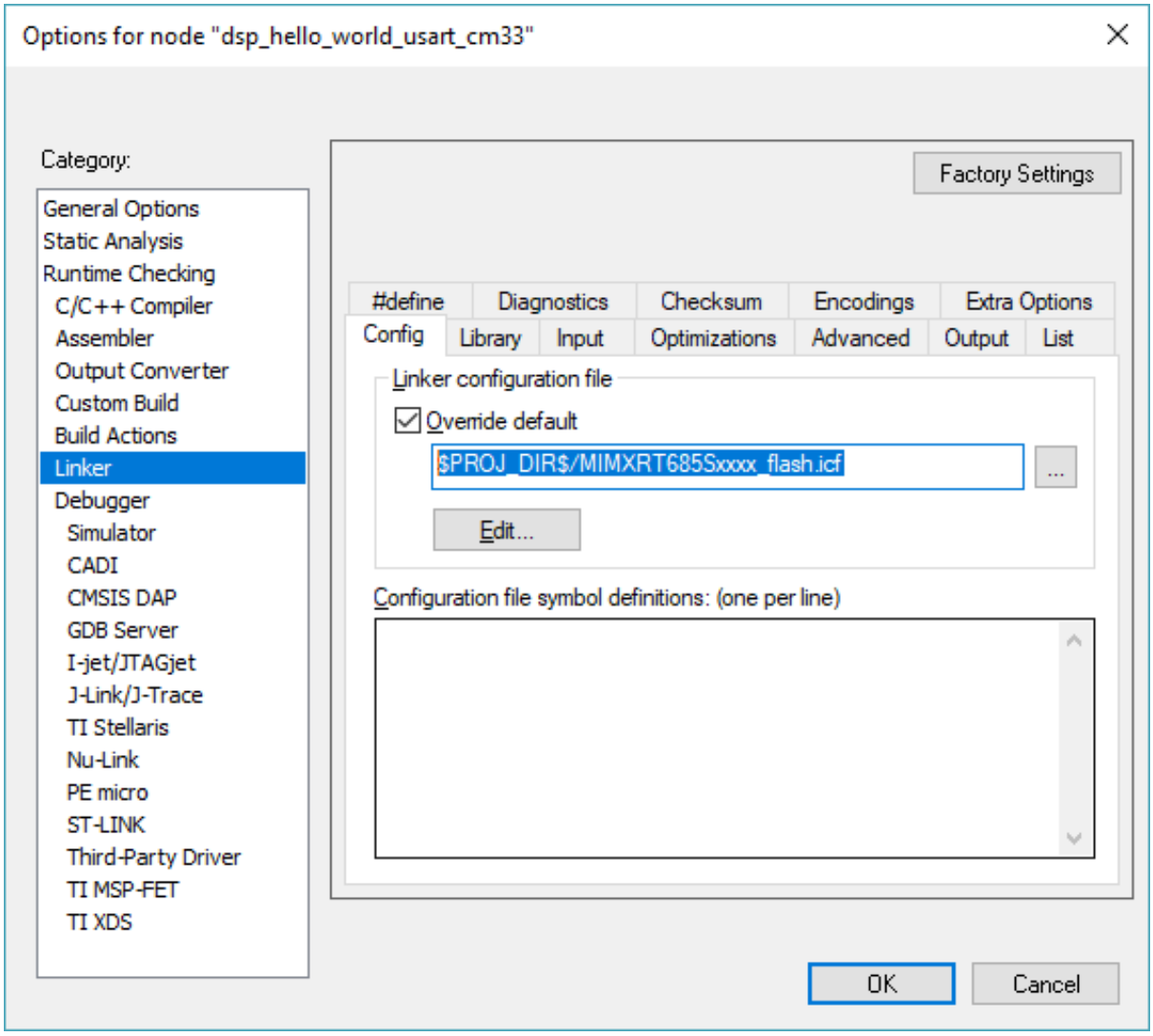
Note that the XHAL call sets cache attribute of the whole region/ 512 M bytes even if the set size is passed. This is also one reason why non-cached attribute is set on the overlapping SRAM address and starts from 0x2000 0000. This also distinguished the physical SRAM addresses starting from 0x0000 0000, which is cacheable area for HiFi4.

Data exchange memory partitions are flexible and can be configured as per application's requirement. However, to mitigate the possible AHB arbitration between the two cores, use of the first eight 32 K and following four 64 K memory partitions is recommended. DSP Data TCM is also used as data exchange area for those data have high demand on timing performance. You must avoid accessing same partition at same time for frequent data exchange between two cores. You can keep one core in Sleep or Wait for Interrupt while another core operating, or set up a ping pong data exchange/ DMA such that when one core fills one partition, another core fetches another partition, and conversely.

For more details about the RT6xx memory map, see the user manual, section 2 Memory Map, and section 2.1.11 HiFi4 memory map.

Boot or Run from Flash

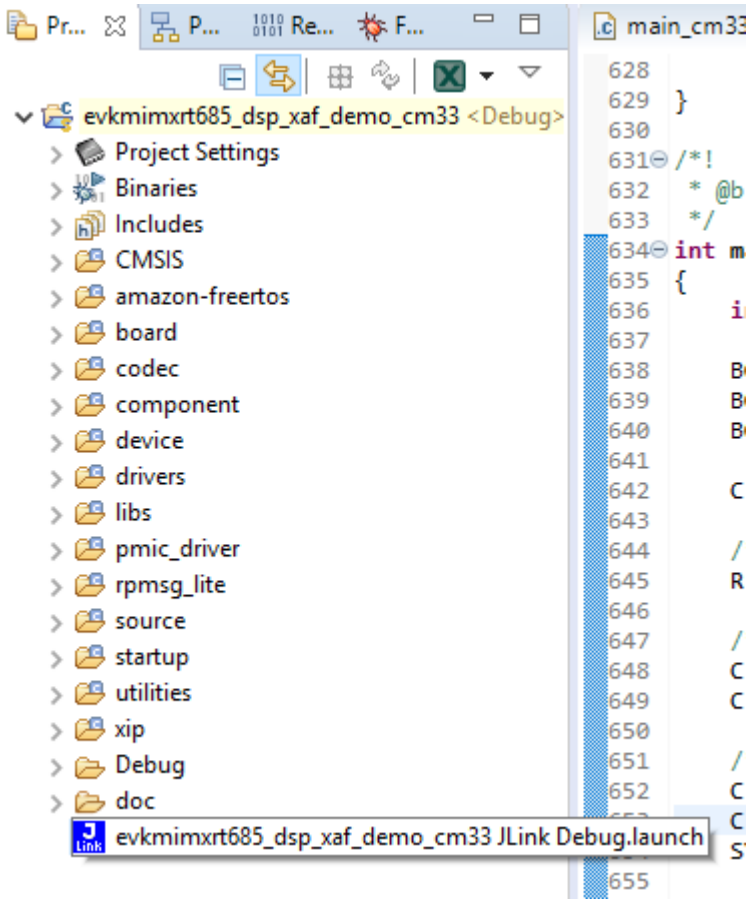
Boot from Flash is straight forward when using IAR environment. SDK provides either two or four different build configurations based on the project: debug (from SRAM)/ release (from SRAM)/ flash_debug/ flash_release. Flash configurations use different memory map in project linker options. For details, see *Figure 1*.



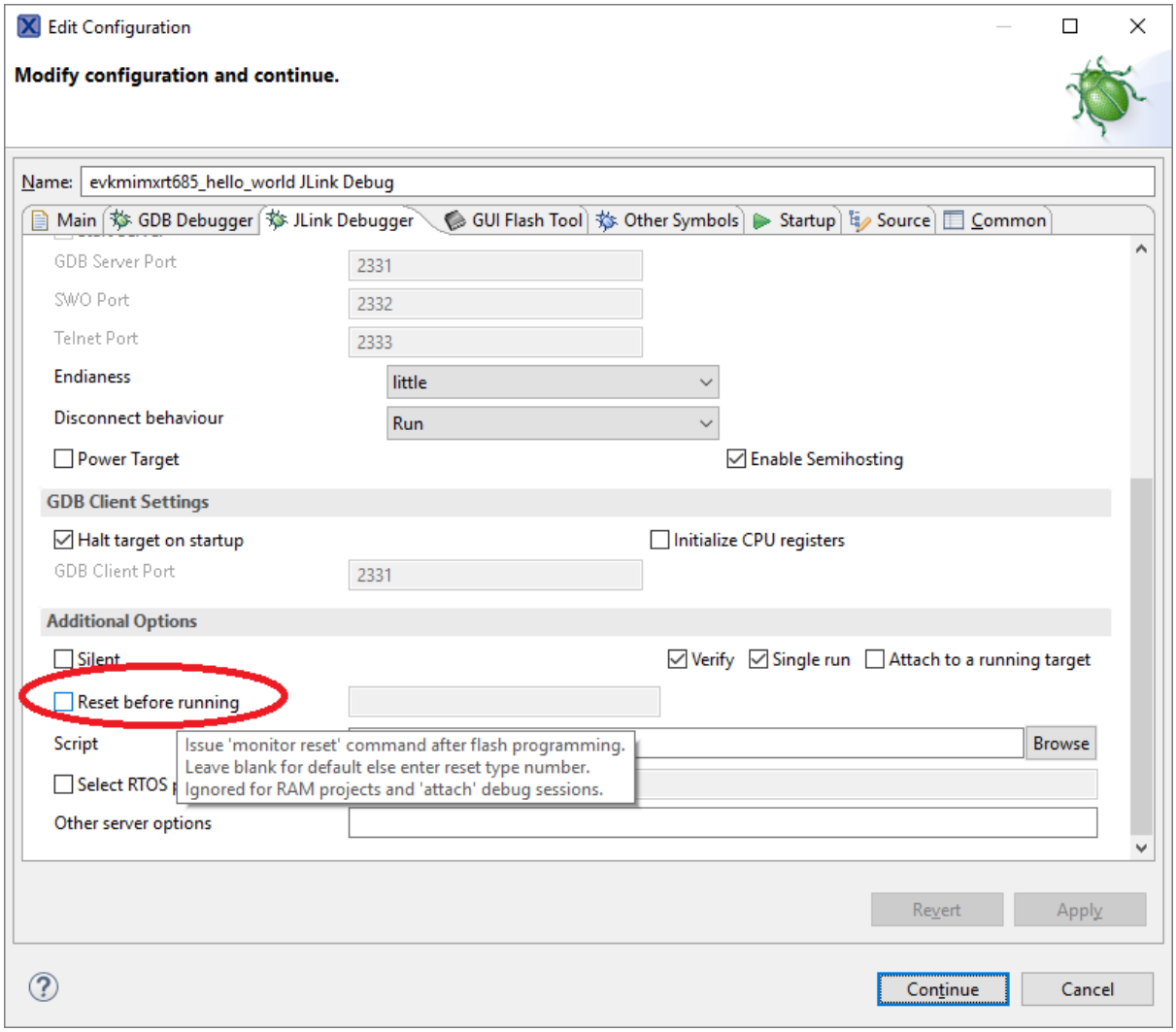
To enable booting from flash, change the ISP mode/ SW5 switches on the EVK.

If using MCUXpresso/ armgcc build environment, note that by default build environment is set to boot from flash. Using MCUXpresso as an example:

1. Make sure to use the latest version of MCUXpresso.
2. Import the SDK examples.
3. Once completed, double click the last file/ J-Link Debug.



4. Launch to modify J-Link debugger setting.
5. Make sure to deselect **Reset before running**. This helps the flash-based program get into main function.



- Make sure to:
  - Modify DSP_IMAGE_COPY_TO_RAM and Define to 1 in **Project settings** > **C/C++ General** > **Paths and Symbols** > **Symbols**.
  - Have the right compilation flag as C/C++. Compilation flags do not work on *.S files.
  - Have #define DSP_IMAGE_COPY_TO_RAM 1 as the first line of source/incbin.S to include the DSP binaries.
  - Use the correct DSP binaries at correct path (must be release binaries).
  - Provide the right image path to incbin.S.
- To run/boot from flash, make sure that the board is set to FlexSPI flash boot mode (ISP2: ISP1: ISP0 = ON, OFF, ON).

Parent topic: HiFi4 Boot Loader and Memory Map

Parent topic:*HiFi4 System Programming*

Peripheral Drivers and Interrupts This section provides information on the basics of peripheral drivers, interrupts, DMA, and how peripheral drivers work on HiFi4 DSP.

Basics

As both cores use shared SRAM and all digital peripherals, SDK drivers work the same on HiFi4 DSP. User Manual section 1.3 Block Diagram shows the system architect on this perspective. For

applications, it is the same to use SDK drivers on HiFi4 DSP with Cortex M33 core. SDK has a DSP hello world UART example showing how easy to use UART on DSP side, same as Cortex M33 programming:

```
#include <xtensa/config/core.h>
#include "fsl_debug_console.h"
    /* Init board hardware. */
    BOARD_InitDebugConsole();
    PRINTF("Hello World running on DSP core '%s', %s_%s\r\n", XCHAL_CORE_ID);
```

Compare with Hello world programming on Cortex M33 side. There are two major differences:

- Pin initialization is not necessary for DSP. It is doable but it is highly recommended managing all pins from Cortex M33 side to make pins all-in-one place. It is also to avoid possible conflicts when you set pins on two different cores;
- Clock setting is not necessary for DSP. It is also doable but same it is highly recommended managing all clock sources all-in-one place;

DMA

RT6xx has two DMA controllers. Each has the same DMA request and trigger input possibilities. The two intended scenarios for their use are:

- One DMA controller (DMA0) is used by the CM33, the other (DMA1) is used by the HiFi4. This case can apply to systems where there is no need to differentiate security between the CPUs or between different tasks.
- One DMA controller (DMA0) is secured and has access to secure spaces and peripherals. The other (DMA1) is not secured and does not have access to secure spaces and peripherals. In this scenario, only the secure code running on the CM33 has access to the secure DMA controller. The other code and the HiFi4 share the non-secure DMA controller (if needed).

HiFi4 DSP always uses DMA1 controller. Again, it is the same to call DMA drivers at HiFi4 DSP side. The DMA destination buffer and DMA descriptor have to be in non-cached area to ensure that each transaction flushes to memory immediately. The below example code showing how to create a DMIC DMA channel:

```
AT_NONCACHEABLE_SECTION_ALIGN(
  static uint8_t s_buffer[BUFFER_SIZE * BUFFER_NUM], 4
);
AT_NONCACHEABLE_SECTION_ALIGN(
dma_descriptor_t s_dmaDescriptorPingpong[2], 16
);
#define DEMO_DMA (DMA1)
#define DEMO_DMIC_RX_CHANNEL DMAREQ_DMIC0
DMA_Init(DEMO_DMA);
DMA_EnableChannel(DEMO_DMA, DEMO_DMIC_RX_CHANNEL);
DMA_SetChannelPriority(DEMO_DMA, DEMO_DMIC_RX_CHANNEL, kDMA_ChannelPriority2);
DMA_CreateHandle(&s_dmicRxDmaHandle, DEMO_DMA, DEMO_DMIC_RX_CHANNEL);
```

Macro AT_NONCACHEABLE_SECTION_ALIGN is defined in SDK driver layer and refer to non-cached area specified by memory map.

For more details about DMA, see the User Manual Chapter 11 RT6xx DMA Controller and SDK DMA examples in <SDK path>\boards\evkmimxrt685\driver_examples\dma.

For audio peripherals/ DMIC/ I2S DMA, see <SDK path>\boards\evkmimxrt685\driver_examples\dmic and <SDK path>\boards\evkmimxrt685\driver_examples\i2s.

Interrupts

HiFi4 DSP has total 32 interrupts and 4 interrupt levels. Besides the first five interrupts, the rest interrupt 5 ~ 31 are multiplexed to allow more control over priorities and more general flexibility. To see the full list, see the User Manual section 8.6.3 DSP Interrupt Input Multiplexers.

It only requires few lines of code to enable an interrupt by calling XOS function calls, and peripheral controls are identical as Cortex M33 side. For example, the below code shows how to enable a UTick timer at DSP side:

First, include XOS system header files and libraries to enable necessary XOS functions.

To include libraries, go to Build **Properties** > **Libraries** > **Add libraries** > **Select 'xos' & 'xtutil'** and **OK** to accept.

```
#include <xtensa/config/core.h>
#include <xtensa/xos.h>
```

Set up the UTick callback and delay functions. This part is identical with Cortex M33 side. For Cortex M33 side implementation, see SDK path\\boards\evkmimxrt685\driver_examples\utick

```
#define UTICK_TIME_1S        (1000000UL)
#define EXAMPLE_UTICK         UTICK0
static volatile bool utickExpired;
static void UTickCallback(void)
{
utickExpired = true;
}
static void UTickDelay(uint32_t usec)
{
/* Set the UTICK timer to wake up the device from reduced power mode */
UTICK_SetTick(EXAMPLE_UTICK, kUTICK_Onetime, usec - 1, UTickCallback);
while (!utickExpired)
{
}
utickExpired = false;
}
```

Initialize the XOS and start UTick timer. The XOS function calls are the differences with Cortex M33 side:

```
/* Initialize XOS thread and start scheduler. Priority 7*/
xos_start_main("main", 7, 0);
/* Init board hardware. */
CLOCK_AttachClk(kLPOSC_to_UTICK_CLK);
CLOCK_EnableClock(kCLOCK_InputMux);
UTICK_Init(EXAMPLE_UTICK);
INPUTMUX_AttachSignal(INPUTMUX, 10U, kINPUTMUX_Utick0ToDspInterrupt);
/* To register interrupt callback */
xos_register_interrupt_handler(15, (XosIntFunc *)UTICK0_DriverIRQHandler, NULL);
/* To enable the interrupt */
xos_interrupt_enable(15);
while (1)
{
UTickDelay(UTICK_TIME_1S);
PRINTF("DSP UTICK TIMER every 1s\r\n");
}
```

Pay attention to the instant numbers. Pick interrupt 15, which maps to DSP_INT0_SEL10 as a L1 interrupts, lowest priority level. For more details about RT6xx HiFi4 DSP interrupt configuration, see User Manual section 51.7 Interrupt. Therefore, for Inputmux, attach UTick interrupt to 10. 15-5=10 as first five interrupts are reserved, not user configurable. To get it to highest priority level, for example L3, configure interrupt as follows.

```
INPUTMUX_AttachSignal(INPUTMUX, 24U,
kINPUTMUX_Utick0ToDspInterrupt);
xos_register_interrupt_handler(29, (XosIntFunc
*)UTICK0_DriverIRQHandler, NULL);
xos_interrupt_enable(29);
```

kINPUTMUX_Utick0ToDspInterrupt specifies DSP interrupt multiplexing value. It has been defined in SDK and matching User Manual section 8.6.3 DSP Interrupt Input Multiplexers.

For more details about XOS interrupt handling, see the Xtensa XOS Reference Manual Chapter 18 Interrupt and Exception Handling, and Chapter 26 Interrupt Handler Restrictions.

Complete Example

To present better how peripheral drivers work on HiFi4 DSP, below list a bare-metal DSP example program that transfers data from DMIC to codec on RT6xx EVKs. The full workspace located in <SDK path>\boards\evkmimxrt685\dsp_examples\audio_demo_bm. This example is derived from Cortex M33 driver <SDK path>\boards\evkmimxrt685\driver_examples\dmic\dmic_i2s_dma with below slightly modifications to adapt to HiFi4 DSP.

- Move DMA buffer and descriptors into non-cached memory partitions;
- Using DMA1 for DSP;
- Calling XOS functions to enable interrupts.
- This example does not contain any pin mux initializing or clock configurations. See the above Cortex M33 example dmic_i2s_dma to set up Cortex M33 side. Once Cortex M33 side ready, this example is compiled and run same as any other SDK DSP examples.

Connect headphone/earphone on audio out of the board, speak on DMIC, or play song nearby the DMIC, you can hear sound on the left channel of headphone/earphone.

```
/* Start XOS */
xos_start_main("main", 7, 0);
/* Disable DSP cache for noncacheable sections. DMA MUST run on none-cacheable/ cache bypass area*/
xthal_set_region_attribute((uint32_t *)&NonCacheable_start,
                    (uint32_t)&NonCacheable_end - (uint32_t)&NonCacheable_start, XCHAL_CA_
↪BYPASS, 0);
xthal_set_region_attribute((uint32_t *)&NonCacheable_init_start,
                    (uint32_t)&NonCacheable_init_end - (uint32_t)&NonCacheable_init_start, XCHAL_
↪CA_BYPASS,
                    0);
PRINTF("Configure DMA\r\n");
/* DSP_INT0_SEL18 = DMA1 */
INPUTMUX_AttachSignal(INPUTMUX, 18U, kINPUTMUX_Dmac1ToDspInterrupt);
/* Map DMA IRQ handler to INPUTMUX selection DSP_INT0_SEL18
 * EXTINT19 = DSP INT 23 */
xos_register_interrupt_handler(XCHAL_EXTINT19_NUM, (XosIntFunc *)DMA_IRQHandle, DMA1);
xos_interrupt_enable(XCHAL_EXTINT19_NUM);
/* The rest DMA & DMIC operations are identical with CM33 side */
```

Parent topic:*HiFi4 System Programming*

Messaging Unit, Semaphore, and IPC Message Unit/ MU and Semaphore/ SEMA42 are essential for DSP programming. The inter-core communications a.k.a IPC on RT600 is based on MU and SEMA42. SDK provides three simple examples about standalone MU and SEMA 42. SDK path \boards\evkmimxrt685\dsp_examples\mu_interrupt, mu_polling, and sema42. These bare-metal examples show how IPC work between two cores. Furthermore, the audio framework demo/ SDK path \boards\evkmimxrt685\dsp_examples\xaf_demo is complete. It uses rpmsg_lite as IPC protocol, which based on MU to transfer messages between the two cores.

First Cortex M33 side initializes rpmsg master, main_cm33.c in app_task()

```
g_my_rpmsg = rpmsg_lite_master_init((void *)RPMSG_LITE_SHMEM_BASE, RPMSG_LITE_
↪SHMEM_SIZE, RPMSG_LITE_LINK_ID, RL_NO_FLAGS);
 g_my_queue = rpmsg_queue_create(g_my_rpmsg);
 g_my_ept   = rpmsg_lite_create_ept(g_my_rpmsg, LOCAL_EPT_ADDR, rpmsg_queue_rx_cb, g_my_
↪queue);
```

Rpmsg initialize MU interrupts for MUA/ master in rpmsg_platform.c platform_init_interrupt()

```
/* Register ISR to environment layer */
env_register_isr(vector_id, isr_data);
env_lock_mutex(platform_lock);
RL_ASSERT(0 <= isr_counter);
if (isr_counter == 0)
{
    MU_EnableInterrupts(APP_MU, (1UL << 27U) >> RPMSG_MU_CHANNEL);
```

For DSP side, it also initializes rpmsg client and tries to hook with the master, in xaf_main_dsp.c:DSP_Main()

```
/* Initialize standard SDK demo application pins */
my_rpmsg = rpmsg_lite_remote_init((void *)RPMSG_LITE_SHMEM_BASE, RPMSG_LITE_LINK_ID,
↪ RL_NO_FLAGS, &rpmsg_ctxt);
while (!rpmsg_lite_is_link_up(my_rpmsg))
{
```

It has rpmsg porting as well, initialize MU interrupts for MUB/ client in rpmsg_platform.c platform_init_interrupt() and enable it in platform_interrupt_enable()

```
xos_register_interrupt_handler(6, MU_B_IRQHandler, ((void *)0));
xos_interrupt_enable(6);
```

Once they hooked up both sides are ready for message exchange. SDK defines various SRTM message structures to pass the commands/ messages. In SDK example, it shows how to pass an MP3 decoding message, a AAC decoding message, a DMIC recording message and so on. Using MP3 decoding message as an example, It fills SRTM structure with input buffer address pointer, input buffer size, output buffer address pointer, output buffer size and so on. And then calling rpmsg to send it to DSP and waiting for the response.

```
rpmsg_lite_send(g_my_rpmsg, g_my_ept, g_remote_addr, (char *)msg, sizeof(THE_MESSAGE), RL_
↪BLOCK);
rpmsg_queue_recv(g_my_rpmsg, g_my_queue, (unsigned long *)&g_remote_addr, (char *)msg,␣
↪sizeof(THE_MESSAGE), len,
                 RL_BLOCK);
```

DSP side listens to rpmsg event.

```
my_ept = rpmsg_lite_create_ept(my_rpmsg, DSP_EPT_ADDR, my_ept_read_cb, (**void** *)&rpmsg_
↪user_data, &my_ept_context);
```

Once received, it handles message and proceeds the command. For MP3 decoding case, it decodes the MP3 data in the input buffer, and flushes the output buffer. It also fills out the SRTM message structure with actual read data size and actual write data size, and eventually sends the response message back to Cortex M33 side.

```
/*Send response message*
   /rpmsg_lite_send(my_rpmsg, my_ept, remote_addr, (**char** *)&msg, **sizeof**(THE_MESSAGE),␣
↪RL_DONT_BLOCK);
```

For this example, DSP is considered as a coprocessor and rpmsg has been used for a light-weight IPC for both cores. All modules are open-sourced and could be easily adapted to whatever application needs.

NatureDSP Library To facilitate application development on RT6xx HiFi4 DSP, NXP licensed NatureDSP signal processing library and embedded as is in source code format. It is found at <SDK path>\middleware\dsp\naturedsp_hifi4.

This is an extensive library, containing the most commonly used signal processing functions: FFT, FIR, vector, matrix, and common mathematics. API and programing guide in.\doc\NatureDSP_Signal_Library_Reference_HiFi4.pdf, and performance data in .\doc\NatureDSP_Signal_Library_Performance_HiFi4.pdf

As this is a huge library, it is impossible to build an all-in-one example on the RT6xx hardware. Fortunately, this library is in source code format and each function/ filter are wrapped in standalone source file. They could be integrated to any application as needed.

System Optimization Performance and power efficiency are key for embedded systems. The following sections list some tips and best practices for system optimization from this perspective.

Profiling

The Xtensa Xplorer IDE tool can run software simulation and profile the application directly. Both simulation and profiling are cycle accurate. This is convenient for algorithm or heavy application developers.

Figure 1 shows profiling result of the helloworld program on simulation console.

```
Events                          Number  Number
                                        per 100
                                        instrs

Committed instructions            3492 ( 100.00 )
Instruction fetches               4066 ( 116.44 )
   From IRAM                      2545 (  72.88 )
   ICache fetches                 1521 (  43.56 )
      ICache misses                 43 (   1.23 )  2.83% of ICache fetches
   Loop buffer hits                338 (   9.68 )  8.31% of Instruction fetches
Taken branches                     702 (  20.10 )
Exceptions                          35 (   1.00 )
   WindowUnderflow                  17 (   0.49 )
   WindowOverflow                   18 (   0.52 )
Loads                              481 (  13.77 )
   From IRAM                        16 (   0.46 )
   DCache loads                    465 (  13.32 )
      DCache load misses             9 (   0.26 )  1.94% of DCache loads
Stores                             308 (   8.82 )
   DCache stores                   308 (   8.82 )
      DCache store misses           10 (   0.29 )  3.25% of DCache stores
DCache castouts                    15 (   0.43 )
PIF transfers (16 bytes each)    1552 (  44.44 )
   IFetch reads                    496 (  14.20 )
   Data reads                      304 (   8.71 )
   Data writes                     240 (   6.87 )
   ICache prefetches               416 (  11.91 )
   DCache prefetches                96 (   2.75 )
ICache prefetch hits                12 (   0.34 ) 27.91% of ICache allocates

Cycles: total = 9078
                                                Summed |           Summed
                                  CPI            CPI   |% Cycle  % Cycle
Committed instructions          3492 ( 1.0000   1.0000 |   38.47    38.47 )
Taken branches                  2121 ( 0.6074   1.6074 |   23.36    61.83 )
Pipeline interlocks              357 ( 0.1022   1.7096 |    3.93    65.76 )
ICache misses                    519 ( 0.1486   1.8582 |    5.72    71.48 )
```

Figure 1 shows the profiling chart, partially.

Profile (Cycles) | Call-Graph | Graph | Saved Output | Pipeline | Dynamic ISA Profile | Console

Function Name	Function (%)	Function (F)	Children (C)	Total (F+C)	Called	Size (bytes)
<TOTAL>	100.00	9,078	N/A	N/A	N/A	N/A
_ResetHandler	45.96	4,173	0	4,173	<spontaneous>	246
xthal_dcache_all_writeback	9.77	887	0	887	1	25
_vfprintf_r	4.40	400	1,776	2,176	1	8,306
_malloc_r	3.39	308	101	409	1	1,318
_WindowUnderflow8	2.61	237	0	237	16	29
_WindowOverflow8	2.48	226	0	226	16	29
_fclose_r	2.40	218	501	719	3	162
__sinit	2.03	185	170	355	1	267
_start	1.78	162	4,688	4,850	1	150
__sfvwrite_r	1.56	142	385	527	1	951
memset	1.56	142	0	142	3	143
__clibrary_init	1.50	137	165	302	1	35
_fflush_r	1.48	135	310	445	4	41
__sflush_r	1.46	133	127	260	4	377

The generated license file only supports debug/run on the RT6xx device target. It does not support software simulation/Xplorer ISS. If there are special requirements to run the software sim-

ulations, contact Cadence directly.

It is also common to measure exact cycle counts for specific processing/ timing measurements. The following is an example code to show how to do the cycle counts.

```
/* Cycle counts inline function */
static unsigned long inline get_ccount(void)
{
    unsigned long r;
    ___asm___ volatile ("rsr.ccount %0" : "=r" (r));
    return r;
}
tic = get_ccount();
processing_function();
toc = get_ccount();
printf("Processing takes %d cycles \r\n", toc - tic);
```

Parent topic: System Optimization

Using Local Memories

RT6xx HiFi4 DSP has 64 K data TCM and 64 K instruction TCM. The TCM is filled with less than 2 K of kernel vectors and the rest is available for application needs. They are the fastest RAM available with no access latency, and can improve critical data/ instruction performance considerably. Consider using TCM as much as possible.

To program code/ data to TCM area:

1. Define macros for sections in TCM. Reuse existing .drm0 and .iram0 sections. Both sections are default TCM sections for every Xplorer project/ memory map. The following are the sections in fsl_common.h.

   ```
   #define DRAM0_DATA ___attribute___((section(".dram0.data")))
   #define DRAM0_BSS ___attribute___((section(".dram0.bss")))
   #define IRAM0_TEXT ___attribute___((section(".iram0.text")))
   #define ALIGNED(alignbytes) ___attribute___((aligned(alignbytes)))
   ```

2. Use macros to declare code and data to be placed in TCM. The data and bss/ uninitialized data are different sections. The following is an example for an FFT function call.

   ```
   DRAM0_DATA const static int32_t fft_in_ref[FFT_LENGTH] = {
   DRAM0_DATA const static int32_t fft_out_ref[FFT_LENGTH] = {
   DRAM0_BSS static int32_t fft_out[FFT_LENGTH];
   IRAM0_TEXT int TEST_FFT()
   {
     ...
         fft_cplx32x32(fft_out, fft_in, FFT_HANDLE, 3);
     ...
   }
   ```

3. The TCM addresses start from 0x2400 0000, which is too far away to main(). Therefore, the project enables long calls to ensure that main() is able to call sub functions. To enable, select **Build Properties** > **Optimization** > **Enable long calls** and select the **Yes** checkbox. Alternatively, add -mlongcalls to the compiler flags.

Power Efficiency

RT6xx HiFi4 DSP has been equipped as a powerful processing core that can run at 600 MHz, but it may not be necessary at all time. It is always recommended to optimize the power efficiency at system level.

Voltage level and core frequency have huge impact on the power efficiency. Using FFT processing as example, continuous FFT may take ~130 mW at Vddcore 1.1 V / 452 MHz, and ~4 mW at

Vddcore 0.7 V / 29 MHz, at room temperature. Profile or do cycle counts for software workload. If it only requires 300 MHz at peak, then the run at full power is not required. Instead, it can run with Vddcore 0.8 V 300 MHz. For more information on the Vddcore and DSP frequency operating conditions, see any *dsp example\cm33\pmic_support.c BOARD_SetPmicVoltageForFreq()*, or see the data sheet, section 13.1 General Operating Conditions.

Some other tips for better power efficiency:

- Turn off DSP/ set DSPSTALL, if needed.
- If possible, make DSP clock adapts to its workload.
- Call XT_WAITI when DSP is in while loop waiting for interrupt. Similar to Arm side __WFI(), it suspends some processor operations to reduce the power consumption.

```
#include <xtensa/tie/xt_interrupt.h >
 extern void XT_WAITI(immediate s);
```

  The immediate value passed is the interrupt level or lower to be IGNORED. For example, if you call XT_WAITI(2) both L1 and L2 interrupts are ignored and only L3 interrupts can wake up DSP.

  Current can further reduce when clocks are turned off to unused memory partitions. For more information, see User Manual 4.5.5.13 DSP SRAM access disable, Register SYSCTL0_DSP_SRAM_ACCESS_DISABLE.

- PLLs consume power. Consider FFRO for low-power use cases.

Parent topic:*HiFi4 System Programming*

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.

1.6 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

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		
EVK-MIMXRT685	MIMXRT633SFAWBR, MIMXRT685SFAWBR, MIMXRT685SVFVKB	MIMXRT633SFFOB, MIMXRT685SFFOB,	MIMXRT633SFVKB, **MIMXRT685SFVKB**,

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

EdgeFast Open Bluetooth EdgeFast Open Bluetooth is NXP's Bluetooth middleware for MCUXpresso SDK. It is based on the Zephyr Bluetooth Host stack and ported to run on FreeRTOS, providing a Zephyr-like Bluetooth API for MCUXpresso SDK applications.

Ethermind BT/BLE Stack nxp_bt_ble_stack

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.

PSA Test Suite Arm Platform Security Architecture Test Suite

mbedTLS mbedtls SSL/TLS library v3.x

wpa_supplicant-rtos NXP Wi-Fi WPA Supplicant

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

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

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 v2.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

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.

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

cJSON Ultralightweight JSON parser in ANSI C

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

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.

New Project Wizard compile failure

The following components request the user to manually select other components that they depend upon in order to compile.

These components depend on several other components and the New Project Wizard (NPW) is not able to decide which one is needed by the user.

Note: xxx means core variants, such as, cm0plus, cm33, cm4, cm33_nodsp.

Components:issdk_mag3110, issdk_host, systick, gpio_kinetis, gpio_lpc, issdk_mpl3115, sensor_fusion_agm01, sensor_fusion_agm01_lpc, issdk_mma845x, issdk_mma8491q, issdk_mma865x, issdk_mma9553, and CMSIS_RTOS2.CMSIS_RTOS2, and components which include cache driver, such as enet_qos.

Also for low-level adapter components, currently the different types of the same adapter cannot be selected at the same time.

For example, if there are two types of timer adapters, gpt_adapter and pit_adapter, only one can be selected as timer adapter

in one project at a time. Duplicate implementation of the function results in an error.

Note: Most of middleware components have complex dependencies and are not fully supported in new project wizard. Adding a middleware component may result in compile failure.

CMSIS PACK new project compile failure

The generated configuration cannot be applied globally. The components, serial_manager_usb_cdc_virtual and serial_manager_usb_cdc_virtual_xxx (xxx means core variants like cm0plus, cm33, cm4, and cm33_nodsp) are unsupported for new project wizard of CMSIS pack and will lead to compile failure if selected while creating new project(s).

Low speed devices not supported

The host examples cannot support low-speed devices.

IAR cannot debug RAM application with J-Link

Currently, IAR will call J-Link reset after the application is downloaded to SRAM, but such operation will cause SRAM data lost.

Here is a workaround to avoid real reset, with the cost of no any reset during the debugging, and hardware status uncleared.

1. Build and debug IAR project once and see the settings folder created.
2. Create the _.JLinkScript file in the settings folder with the following contents.

```
void ResetTarget(void) {
JLINK_TARGET_Halt();
}
```

3. Debug the project again and now it can work.

lwip_httpssrv_ota_wifi example fails to accept the new image on EVKMIMXRT685

After uploading the new image over https, and rebooting the board the new image starts. However, reboot starts it is not possible to accept the update and make it permanent. This issue is specific to combination of EVKMIMXRT685 board and AW-NM191NF-uSD Wi-Fi module. Other configurations are not affected.

Board may reset itself when running SD card related cases

Board may reset itself when running SD card related cases. However, the issue is not reproduced if more power supply is connected.

Log output may be mixed in shell/hfp example

When multiple tasks print the log, the serial port terminal output has the probability to appear mixed.

LE encryption failure causes connection to fail

There can be a corner case when LE link encryption can fail. This occurs when device under test (DUT); RT Bluetooth controller here, while waiting for the response to LL_SLAVE_FEATURES_REQ, instead receives the LL_ENC_REQ response from a remote device.

This causes deadlock scenario where DUT and remote devices are stuck waiting for response from peer.

Connection disconnects with 7.5 ms connection interval

When wireless example works as a peripheral, the central role with 7. 5 ms connection interval connects to the wireless example. However, when using the 6th/7th/8th central to connect the wireless example, all the previous connections with the previous centrals are disconnected.

When wireless example works as central and the connection interval is 7.5 ms, the 4th peripheral is not scanned.

a2dp sink demo: Noise may occur when phone plays music with other operations

Noise may occur when phone plays music with other operations.

For example:

- Switches the WIFI network of the phone when playing music
- Switches music when playing the online music

Examples hello_world_ns, secure_faults_ns, and secure_faults_trdc_ns have incorrect library path in GUI projects

When the affected examples are generated as GUI projects, the library linking the secure and non-secure worlds has an incorrect path set. This causes linking errors during project compilation.

Examples: hello_world_ns, hello_world_s, secure_faults_ns, secure_faults_s, secure_faults_trdc_ns, secure_faults_trdc_s

Affected toolchains: mdk, iar

Workaround: In the IDE project settings for the non-secure (_ns) project, find the linked library (named hello_world_s_CMSE_lib.o, or similar, depending on the example project) and replace the path to the library with <build_directory>/<secure_world_project_folder>/<IDE>/, replacing the subdirectory names with the build directory, the secure world project name, and IDE name.

1.7 ChangeLog

1.7.1 MCUXpresso SDK Changelog

Board Support Files

board [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.

CASPER

[2.2.5]

- Fix Coverity Uninitialized scalar variable.

[2.2.4]

- Fix MISRA-C 2012 issue.

[2.2.3]

- Added macro into CASPER_Init and CASPER_Deinit to support devices without clock and reset control.

[2.2.2]

- Enable hardware interleaving to RAMX0 and RAMX1 for CASPER by feature macro FSL_FEATURE_CASPER_RAM_HW_INTERLEAVE

[2.2.1]

- Fix MISRA C-2012 issue.

[2.2.0]

- Rework driver to support multiple curves at once.

[2.1.0]

- Add ECC NIST P-521 elliptic curve.

[2.0.10]

- Fix MISRA C-2012 issue.

[2.0.9]

- Remove unused function Jac_oncurve().
- Fix ECC384 build.

[2.0.8]

- Add feature macro for CASPER_RAM_OFFSET.

[2.0.7]

- Fix MISRA C-2012 issue.

[2.0.6]

- Bug Fixes
  - Fix IAR Pa082 warning

[2.0.5]

- Bug Fixes
  - Fix sign-compare warning

[2.0.4]

- For GCC compiler, enforce O1 optimize level, specifically to remove strict-aliasing option. This driver is very specific and requires -fno-strict-aliasing.

[2.0.3]

- Bug Fixes
  - Fixed the bug for KPSDK-28107 RSUB, FILL and ZERO operations not implemented in enum _casper_operation.

[2.0.2]

- Bug Fixes
  - Fixed KPSDK-25015 CASPER_MEMCPY hard-fault on LPC55xx when both source and destination buffers are outside of CASPER_RAM.

[2.0.1]

- Bug Fixes
  - Fixed the bug that KPSDK-24531 double_scalar_multiplication() result may be all zeroes for some specific input.

[2.0.0]

- Initial version.

CLOCK

[2.7.4]

- New feature
  - Added ESPI clock selection.

[2.7.3]

- Improvements
  - Added return value for CLOCK_InitSysPfd CLOCK_InitAudioPfd.

[2.7.2]

- Bug Fixes
  - Added clock name array for PUF, HashCrypt and Casper.

[2.7.1]

- Improvements
  - Added lost comments for some enumerations.

[2.7.0]

- Bug Fixes
  - Updated enum sys_pll_mult_t and audio_pll_mult_t to fix the supported MULT values for PLLs.
  - Supported more APIs for DSP core.

[2.6.1]

- New feature
  - Added CLOCK_SetClkinFreq API.
- Other Changes
  - Rmove main_clk from UTICK,WWDT0/1 clock attach define.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rules.

[2.6.0]

- API change
  - Added enableLowPower parameter in CLOCK_EnableSysOscClk().
  - Added id parameter in CLOCK_GetSdioClkFreq().
- Other Changes
  - Fixed C++ build errors.
  - Added sdio1 and flexcomm6,7 clock support.
  - Added syspll pfd clock dividers define.

  - Updated register access per the B0 header file's change.
  - Added assert in CLOCK_SetFRGClock(), the FRG DIV should be always set to 0xFF according to User Manual.

[2.5.0]

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

[2.4.1]

- Bug Fixes
  - Avoided waiting REQFLAG when divider configured to HALT in CLOCK_SetClkDiv().

[2.4.0]

- Modify Audio PLL mult parameter range.
- Update get mclk_in api.

[2.3.0]

- New feature
  - Adding Deinit PLL&PFD API.
- API change
  - Add delay_us parameter in CLOCK_EnableSysOscClk()

[2.2.0]

- Update I3C and SDIO clock get API.
- Modify wwdt clock API name.

[2.1.1]

- Update PLL initialization function.

[2.1.0]

- New feature
  - Adding new API CLOCK_DelayAtLeastUs() implemented by DWT to allow users set delay in unit of microsecond.

[2.0.5]

- Clean up 3 USB clock enable APIs;

[2.0.4]

- New feature
  - add get mclk_in api.
- Bug Fixes
  - The Systick clock value should not be divided except choose main clock.

[2.0.3]

- Update according to UM 0.7.

[2.0.2]

- some minor fixes.

[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

[2.1.1]

- Fix MISRA issue.

[2.1.0]

- Add CRC_WriteSeed function.

[2.0.2]

- Fix MISRA issue.

[2.0.1]

- Fixed KPSDK-13362. MDK compiler issue when writing to WR_DATA with -O3 optimize for time.

[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.

LPC_DMA

[2.5.5]

- New Features
  - Added common IRQ handler entry DMA_DriverIRQHandler.
- Bug Fixes
  - Fixed CERT INT30-C violations.

[2.5.4]

- Bug Fixes
  - Fixed coverity issues with CERT INT30-C, CERT INT31-C compliance.

[2.5.3]

- Improvements
  - Add assert in DMA_SetChannelXferConfig to prevent XFERCOUNT value overflow.

[2.5.2]

- Bug Fixes
  - Use separate “SET” and “CLR” registers to modify shared registers for all channels, in case of thread-safe issue.

[2.5.1]

- Bug Fixes
  - Fixed violation of the MISRA C-2012 rule 11.6.

[2.5.0]

- Improvements
  - Added a new api DMA_SetChannelXferConfig to set DMA xfer config.

[2.4.4]

- Bug Fixes
  - Fixed the issue that DMA_IRQHandle might generate redundant callbacks.
  - Fixed the issue that DMA driver cannot support channel bigger then 32.
  - Fixed violation of the MISRA C-2012 rule 13.5.

[2.4.3]

- Improvements
  - Added features FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZEn/FSL_FEATURE_DMA0_DESCRIPTOR_A to support the descriptor align size not constant in the two instances.

[2.4.2]

- Bug Fixes
  - Fixed violation of the MISRA C-2012 rule 8.4.

[2.4.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 5.7, 8.3.

[2.4.0]

- Improvements
  - Added new APIs DMA_LoadChannelDescriptor/DMA_ChannelIsBusy to support polling transfer case.
- Bug Fixes
  - Added address alignment check for descriptor source and destination address.
  - Added DMA_ALLOCATE_DATA_TRANSFER_BUFFER for application buffer allocation.
  - Fixed the sign-compare warning.
  - Fixed violations of the MISRA C-2012 rules 18.1, 10.4, 11.6, 10.7, 14.4, 16.3, 20.7, 10.8, 16.1, 17.7, 10.3, 3.1, 18.1.

[2.3.0]

- Bug Fixes
  - Removed DMA_HandleIRQ prototype definition from header file.
  - Added DMA_IRQHandle prototype definition in header file.

[2.2.5]

- Improvements
  - Added new API DMA_SetupChannelDescriptor to support configuring wrap descriptor.
  - Added wrap support in function DMA_SubmitChannelTransfer.

[2.2.4]

- Bug Fixes
  - Fixed the issue that macro DMA_CHANNEL_CFER used wrong parameter to calculate DSTINC.

[2.2.3]

- Bug Fixes
  - Improved DMA driver Deinit function for correct logic order.
- Improvements
  - Added API DMA_SubmitChannelTransferParameter to support creating head descriptor directly.
  - Added API DMA_SubmitChannelDescriptor to support ping pong transfer.
  - Added macro DMA_ALLOCATE_HEAD_DESCRIPTOR/DMA_ALLOCATE_LINK_DESCRIPTOR to simplify DMA descriptor allocation.

[2.2.2]

- Bug Fixes
  - Do not use software trigger when hardware trigger is enabled.

[2.2.1]

- Bug Fixes
  - Fixed Coverity issue.

[2.2.0]

- Improvements
  - Changed API DMA_SetupDMADescriptor to non-static.
  - Marked APIs below as deprecated.
    - DMA_PrepareTransfer.
    - DMA_Submit transfer.
  - Added new APIs as below:
    - DMA_SetChannelConfig.
    - DMA_PrepareChannelTransfer.
    - DMA_InstallDescriptorMemory.
    - DMA_SubmitChannelTransfer.
    - DMA_SetChannelConfigValid.

    * DMA_DoChannelSoftwareTrigger.
    * DMA_LoadChannelTransferConfig.

[2.0.1]

- Improvements
  - Added volatile for DMA descriptor member xfercfg to avoid optimization.

[2.0.0]

- Initial version.

DMIC

[2.3.4]

- New Features
  - Added common IRQ handler entry DMIC_CommonDriverIRQHandler.

[2.3.3]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8.

[2.3.2]

- New Features
  - Supported 4 channels in driver.

[2.3.1]

- Bug Fixes
  - Fixed the issue that DMIC_EnableChannelDma and DMIC_EnableChannelFifo did not clean relevant bits.

[2.3.0]

- Improvements
  - Added new apis DMIC_ResetChannelDecimator/DMIC_EnableChannelGlobalSync/DMIC_DisableChann

[2.2.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 14.4, 17.7, 10.4, 10.3, 10.8, 14.3.

[2.2.0]

- Bug Fixes
  - Corrected the usage of feature FSL_FEATURE_DMIC_IO_HAS_NO_BYPASS.

[2.1.1]

- Improvements
  - Added feature FSL_FEATURE_DMIC_HAS_NO_IOCFG for IOCFG register.

[2.1.0]

- New Features
  - Added API DMIC_EnbleChannelInterrupt/DMIC_EnbleChannelDma to replace API DMIC_SetOperationMode.
  - Added API DMIC_SetIOCFG and marked DMIC_ConfigIO as deprecated.
  - Added API DMIC_EnableChannelSignExtend to support sign extend feature.

[2.0.5]

- Improvements
  - Changed some parameters' value of DMIC_FifoChannel API, such as enable, resetn, and trig_level. This is not possible for the current code logic, so it improves the DMIC_FifoChannel logic and fixes incorrect math logic.

[2.0.4]

- Bug Fixes
  - Fixed the issue that DMIC DMA driver(ver2.0.3) did not support calling DMIC_TransferReceiveDMA in DMA callback as it did before version 2.0.3. But calling DMIC_TransferReceiveDMA in callback is not recommended.

[2.0.3]

- New Features
- Supported linked transfer in DMIC DMA driver.
- Added new API DMIC_EnableChannelFifo/DMIC_DoFifoReset/DMIC_InstallDMADescriptor.

[2.0.2]

- New Features
  - Supported more channels in driver.

[2.0.1]

- New Features
  - Added a control macro to enable/disable the RESET and CLOCK code in current driver.

[2.0.0]

- Initial version.

DMIC_DMA

[2.4.2]

- Bug Fixes
  - Fixed coverity High Impact finding

[2.4.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8.

[2.4.0]

- Bug Fixes
  - Fixed the issue that DMIC_TransferAbortReceiveDMA can not disable dmic and dma request issue.

[2.3.1]

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

[2.3.0]

- Refer DMIC driver change log 2.0.1 to 2.3.0

DSP

[2.1.1]

- Fixed Misra issue.

[2.1.0]

- Allowed multiple calls to DSP_Init.
- Removed DSP clock gate operation.

[2.0.1]

- Update DSP Init&Deinit function.

[2.0.0]

- initial version.

FLEXCOMM

[2.0.3]

- New Features
  - Added common IRQ handler entry FLEXCOMM_DriverIRQHandler.

[2.0.2]

- Bug Fixes
  - Fixed typos in FLEXCOMM15_DriverIRQHandler().
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 10.7, 10.8, 11.3, 11.6, 11.8, 11.9, 13.5.
- Improvements
  - Added instance calculation in FLEXCOMM16_DriverIRQHandler() to align with Flexcomm 14 and 15.

[2.0.1]

- Improvements
  - Added more IRQHandler code in drivers to adapt new devices.

[2.0.0]

- Initial version.

FLEXSPI

[2.9.3]

- Bug Fixes
  - Fixed CERT INT30-C violations in FLEXSPI_TransferBlocking: added assert guards before unsigned arithmetic on flashAddress to prevent wrap-around.
  - Fixed CERT INT31-C violations in FLEXSPI_Init and FLEXSPI_UpdateAhbBuffersSettings: added assert to verify FSL_FEATURE_FLEXSPI_AHB_RX_BUFFER_SIZEn returns a non-negative value before casting to uint32_t.

[2.9.2]

- New Features
  - Added common IRQ handler entry FLEXSPI_CommonDriverIRQHandler.

[2.9.1]

- Bug Fixes
  - Fixed CERT INT31-C violations: added masks before uint8_t casts and replaced bool fields with ternary expressions.
- Improvements
  - Modified FLEXSPI_WriteBlocking() to trigger IP command after TX FIFO is filled or when all data has been written, preventing premature command execution
  - Updated $\mathrm{FLEXSPI_TransferBlocking()}$ to distinguish between Write and Read operation flows according to FlexSPI IP command specification.
    - For Write/Program operations: Fill IP transmit FIFO with data before triggering the IP command.
    - For Read operations: Trigger IP command first, then read data from IP receive FIFO.

* Added detailed flow comments for both Write and Read operations.

[2.9.0]

- New Features
  - Added FlexSPI root clock divider configuration.
- Bug Fixes
  - Add explicit boolean to integer conversions (? 1U : 0U) for all enable flags
  - Add assertions to prevent unsigned integer overflow in buffer size calculations
  - Add assertions to prevent unsigned integer underflow in watermark calculations
  - Add safe type conversion for uint32_t to uint8_t operations
  - Add bounds checking assertion for LUT index operations
  - Follow INT30-C and INT31-C CERT coding standards

[2.8.1]

- Improvements
  - Updated the LUT configuration parameter checking with flexible way to adapt different Socs.

[2.8.0]

- Bug Fixes
  - Introduced the **disableAhbReadResume** field in the flexspi_config_t structure to provide control over the AHBCR[RESUMEDISABLE] register bit.
  - Implemented a workaround for hardware erratum ERR052733 by setting the default value of **disableAhbReadResume** to **true**.
  - Fixed issue in FLEXSPI_TransferHandleIRQ where the transfer completion was incorrectly signaled despite pending read/write operations.
- New Features
  - Introduced a new function(FLEXSPI_UpdateAhbBuffersSettings) that allows users to update the AHB buffer configuration after the FLEXSPI module has been initialized

[2.7.0]

- New Features
  - Added new API to support address remapping.

[2.6.4]

- Improvements
  - Added new macro “FSL_SDK_ENABLE_FLEXSPI_RESET_CONTROL” to support driver level reset control.

[2.6.3]

- Bug Fixes
  - Fixed an issue which cause IPCR1[IPAREN] cleared by mistake.

[2.6.2]

- Bug Fixes
  - Wait Bus IDLE before operation of FLEXSPI_SoftwareReset(), FLEXSPI_TransferBlocking() and FLEXSPI_TransferNonBlocking().

[2.6.1]

- Bug Fixes
  - Updated code of reset peripheral.
  - Updated FLEXSPI_UpdateLUT() to check if input lut address is not in Flexspi AMBA region.
  - Updated FLEXSPI_Init() to check if input AHB buffer size exceeded maximum AHB size.

[2.6.0]

- New Features
  - Added new API to set AHB memory-mapped flash base address.
  - Added support of DLLxCR[REFPHASEGAP] bit field, it is recommended to set it as 0x2 if DLL calibration is enabled.

[2.5.1]

- Bugfixes
  - Fixed handling of W1C bits in the INTR register
  - Removed FIFO resets from FLEXSPI_CheckAndClearError
  - FLEXSPI_TransferBlocking is observing IPCMDDONE and then fetches the final status of the transfer
  - Fixed issue that FLEXSPI2_DriverIRQHandler not defined.

[2.5.0]

- Improvements
  - Supported word un-aligned access for write/read blocking/non-blocking API functions.
  - Fixed dead loop issue in DLL update function when using FRO clock source.
  - Fixed violations of the MISRA C-2012 Rule 10.3.

[2.4.0]

- Improvements
  - Isolated IP command parallel mode and AHB command parallel mode using feature MACRO.
  - Supported new column address shift feature for external memory.

[2.3.5]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 14.2.

[2.3.4]

- Bug Fixes
  - Updated flexspi_config_t structure and FlexSPI_Init to support new feature FSL_FEATURE_FLEXSPI_HAS_NO_MCR0_CONBINATION.

[2.3.3]

- Bug Fixes
  - Removed feature FSL_FEATURE_FLEXSPI_DQS_DELAY_PS for DLL delay setting. Changed to use feature FSL_FEATURE_FLEXSPI_DQS_DELAY_MIN to set slave delay target as 0 for DLL enable and clock frequency higher than 100MHz.

[2.3.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 8.4, 8.5, 10.1, 10.3, 10.4, 11.6 and 14.4.

[2.3.1]

- Bug Fixes
  - Wait for bus to be idle before using it as access to external flash with new setting in FLEXSPI_SetFlashConfig() API.
  - Fixed the potential buffer overread and Tx FIFO overwrite issue in FLEXSPI_WriteBlocking.

[2.3.0]

- New Features
  - Added new API FLEXSPI_UpdateDllValue for users to update DLL value after updating flexspi root clock.
  - Corrected grammatical issues for comments.
  - Added support for new feature FSL_FEATURE_FLEXSPI_DQS_DELAY_PS in DLL configuration.

[2.2.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3 and 10.4.
  - Updated _flexspi_command from named enumerator into anonymous enumerator.

[2.2.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3, 10.4, 10.8, 11.9, 14.4, 15.7, 16.4, 17.7, 7.3.
  - Fixed IAR build warning Pe167.
  - Fixed the potential buffer overwrite and Rx FIFO overread issue in FLEXSPI_ReadBlocking.

[2.2.0]

- Bug Fixes
  - Fixed flag name typos: kFLEXSPI_IpTxFifoWatermarkEmpltyFlag to kFLEXSPI_IpTxFifoWatermarkEmptyFlag; kFLEXSPI_IpCommandExcutionDoneFlag to kFLEXSPI_IpCommandExecutionDoneFlag.
  - Fixed comments typos such as sequencen->sequence, levle->level.
  - Fixed FLSHCR2[ARDSEQID] field clean issue.
  - Updated flexspi_config_t structure and FlexSPI_Init to support new feature FSL_FEATURE_FLEXSPI_HAS_NO_MCR0_ATDFEN and FSL_FEATURE_FLEXSPI_HAS_NO_MCR0_ARDFEN.
  - Updated flexspi_flags_t structure to support new feature FSL_FEATURE_FLEXSPI_HAS_INTEN_AHBBUSERROREN.

[2.1.1]

- Improvements
  - Defaulted enable prefetch for AHB RX buffer configuration in FLEXSPI_GetDefaultConfig, which is align with the reset value in AHBRXBUFxCR0.
  - Added software workaround for ERR011377 in FLEXSPI_SetFlashConfig; added some delay after DLL lock status set to ensure correct data read/write.

[2.1.0]

- New Features
  - Added new API FLEXSPI_UpdateRxSampleClock for users to update read sample clock source after initialization.
  - Added reset peripheral operation in FLEXSPI_Init if required.

[2.0.5]

- Bug Fixes
  - Fixed FLEXSPI_UpdateLUT cannot do partial update issue.

[2.0.4]

- Bug Fixes
  - Reset flash size to zero for all ports in FLEXSPI_Init; fixed the possible out-of-range flash access with no error reported.

[2.0.3]

- Bug Fixes
  - Fixed AHB receive buffer size configuration issue. The FLEXSPI_AHBRXBUFCR0_BUFSZ field should configure 64 bits size, and currently the AHB receive buffer size is in bytes which means 8-bit, so the correct configuration should be config->ahbConfig.buffer[i].bufferSize / 8.

[2.0.2]

- New Features
  - Supported DQS write mask enable/disable feature during set FLEXSPI configuration.
  - Provided new API FLEXSPI_TransferUpdateSizeEDMA for users to update eDMA transfer size(SSIZE/DSIZE) per DMA transfer.
- Bug Fixes
  - Fixed invalid operation of FLEXSPI_Init to enable AHB bus Read Access to IP RX FIFO.
  - Fixed incorrect operation of FLEXSPI_Init to configure IP TX FIFO watermark.

[2.0.1]

- Bug Fixes
  - Fixed the flag clear issue and AHB read Command index configuration issue in FLEXSPI_SetFlashConfig.
  - Updated FLEXSPI_UpdateLUT function to update LUT table from any index instead of previous command index.
  - Added bus idle wait in FLEXSPI_SetFlashConfig and FLEXSPI_UpdateLUT to ensure bus is idle before any change to FlexSPI controller.
  - Updated interrupt API FLEXSPI_TransferNonBlocking and interrupt handle flow FLEXSPI_TransferHandleIRQ.
  - Updated eDMA API FLEXSPI_TransferEDMA.

[2.0.0]

- Initial version.

FLEXSPI DMA Driver

[2.2.3]

- Bug Fixes
  - Fixed CERT INT31-C violations in FLEXSPI_WriteDataDMA and FLEXSPI_ReadDataDMA: moved uint8_t cast outside multiplication and added masks before narrowing conversions.
  - Fixed CERT INT30-C violations: added assert guards before power-of-2 checks to prevent unsigned wrap on handle->count - 1U.

[2.2.2]

- Bug Fixes
  - Fixed CERT INT31-C violations: added masks before uint8_t casts and explicit uint32_t casts for division results.

[2.2.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3, 10.4, 10.8.

[2.2.0]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 Rule 10.1, 10.3.
- New Features
  - Updated name of FLEXSPI_TransferGetTransferCountDMA API.

[2.1.1]

- New Features
  - Updated driver to support feature FSL_FEATURE_FLEXSPI_DMA_MULTIPLE_DES.

[2.1.0]

- Bug Fixes
  - Updated enumaration flexspi_dma_transfer_nsize_t and remove the unsupported items.
- New Features
  - Updated driver for deprecating the multiple linked descriptors inside FLEXSPI_TransferDMA, only up to one linked descriptor is needed according to hardware update.

[2.0.0]

- Initial version.

FMEAS

[2.1.2]

- Improvements
  - Replace macros with inline functions for clock count retrieval.
- Bug Fixes
  - Fixed violations of the CERT INT30-C INT31-C.

[2.1.1]

- Bug Fixes
  - MISRA C-2012 issues fixed: rule 10.4, rule 10.8.

[2.1.0]

- Updated "FMEAS_GetFrequency","FMEAS_StartMeasure","FMEAS_IsMeasureComplete" API and add definition to match ASYNC_SYSCON.

[2.0.0]

- Initial version ported from LPCOpen.

GPIO

[2.1.7]

- Improvements
  - Enhanced GPIO_PinInit to enable clock internally.

[2.1.6]

- Bug Fixes
  - Clear bit before set it within GPIO_SetPinInterruptConfig() API.

[2.1.5]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 3.1, 10.6, 10.7, 17.7.

[2.1.4]

- Improvements
  - Added API GPIO_PortGetInterruptStatus to retrieve interrupt status for whole port.
  - Corrected typos in header file.

[2.1.3]

- Improvements
  - Updated "GPIO_PinInit" API. If it has DIRCLR and DIRSET registers, use them at set 1 or clean 0.

[2.1.2]

- Improvements
  - Removed deprecated APIs.

[2.1.1]

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

[2.1.0]

- New Features
  - Added GPIO initialize API.

[2.0.0]

- Initial version.

HASHCRYPT

[2.0.0]

- Initial version.

[2.0.1]

- Supported loading AES key from unaligned address.

[2.0.2]

- Supported loading AES key from unaligned address for different compiler and core variants.

[2.0.3]

- Remove SHA512 and AES ICB algorithm definitions

[2.0.4]

- Add SHA context switch support

[2.1.0]

- Update the register name and macro to align with new header.
- Fixed the sign-compare warning in hashcrypt_load_data.

[2.1.1]

- Fix MISRA C-2012.

[2.1.2]

- Support loading AES input data from unaligned address.

[2.1.3]

- Fix MISRA C-2012.

[2.1.4]

- Fix context switch cannot work when switching from AES.

[2.1.5]

- Add data synchronization barrier inside hashcrypt_sha_ldm_stm_16_words() to prevent possible optimization issue.

[2.2.0]

- Add AES-OFB and AES-CFB mixed IP/SW modes.

[2.2.1]

- Add data synchronization barrier inside hashcrypt_sha_ldm_stm_16_words() prevent compiler from reordering memory write when -O2 or higher is used.

[2.2.2]

- Add data synchronization barrier inside hashcrypt_sha_ldm_stm_16_words() to fix optimization issue

[2.2.3]

- Added check for size in hashcrypt_aes_one_block to prevent overflowing COUNT field in MEMCTRL register, if its bigger than COUNT field do a multiple runs.

[2.2.4]

- In all HASHCRYPT_AES_xx functions have been added setting CTRL_MODE bitfield to 0 after processing data, which decreases power consumption.

[2.2.5]

- Add data synchronization barrier and instruction synchronization barrier inside hashcrypt_sha_process_message_data() to fix optimization issue

[2.2.6]

- Add data synchronization barrier inside HASHCRYPT_SHA_Update() and hashcrypt_get_data() function to fix optimization issue on MDK and ARMGCC release targets

[2.2.7]

- Add data synchronization barrier inside HASHCRYPT_SHA_Update() to fix optimization issue on MCUX IDE release target

[2.2.8]

- Unify hashcrypt hashing behavior between aligned and unaligned input data

[2.2.9]

- Add handling of set ERROR bit in the STATUS register

[2.2.10]

- Fix missing error statement in hashcrypt_save_running_hash()

[2.2.11]

- Fix incorrect SHA-256 calculation for long messages with reload

[2.2.12]

- Fix hardfault issue on the Keil compiler due to unaligned memcpy() input on some optimization levels

[2.2.13]

- Added function hashcrypt_seed_prng() which loading random number into PRNG_SEED register before AES operation for SCA protection

[2.2.14]

- Modify function hashcrypt_get_data() to prevent issue with unaligned access

[2.2.15]

- Add wait on DIGEST BIT inside hashcrypt_sha_one_block() to fix issues with some optimization flags

[2.2.16]

- Add DSB instruction inside hashcrypt_sha_ldm_stm_16_words() to fix issues with some optimization flags

[2.2.17]

- Fix context size when hashcrypt built with reload feature

[2.2.18]

- Rename the internal BUILD_ASSERT macro to HASHCRYPT_BUILD_ASSERT to avoid conflicts with environment-specific implementations.

I2C

[2.3.4]

- Bug Fixes
  - Added internal reset of master function after arbitration lost in I2C_MasterTransferHandleIRQ(), otherwise the STAT[MSTARBLOSS] is not set in next transfer.

[2.3.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1.
  - Fixed issue that if master only sends address without data during I2C interrupt transfer, address nack cannot be detected.

[2.3.2]

- Improvement
  - Enable or disable timeout option according to enableTimeout.
- Bug Fixes
  - Fixed timeout value calculation error.
  - Fixed bug that the interrupt transfer cannot recover from the timeout error.

[2.3.1]

- Improvement
  - Before master transfer with transactional APIs, enable master function while disable slave function and vise versa for slave transfer to avoid the one affecting the other.
- Bug Fixes
  - Fixed bug in I2C_SlaveEnable that the slave enable/disable should not affect the other register bits.

[2.3.0]

- Improvement
  - Added new return codes kStatus_I2C_EventTimeout and kStatus_I2C_SclLowTimeout, and added the check for event timeout and SCL timeout in I2C master transfer.
  - Fixed bug in slave transfer that the address match event should be invoked before not after slave transmit/receive event.

[2.2.0]

- New Features
  - Added enumeration _i2c_status_flags to include all previous master and slave status flags, and added missing status flags.
  - Modified I2C_GetStatusFlags to get all I2C flags.
  - Added API I2C_ClearStatusFlags to clear all clearable flags not just master flags.

  - Modified master transactional APIs to enable bus event timeout interrupt during transfer, to avoid glitch on bus causing transfer hangs indefinitely.
- Bug Fixes
  - Fixed bug that status flags and interrupt enable masks share the same enumerations by adding enumeration _i2c_interrupt_enable for all master and slave interrupt sources.

[2.1.0]

- Bug Fixes
  - Fixed bug that during master transfer, when master is nacked during slave probing or sending subaddress, the return status should be kStatus_I2C_Addr_Nak rather than kStatus_I2C_Nak.
- Bug Fixes
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.4, 13.5.
- New Features
  - Added macro I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK, so that user can configure whether to ignore the last byte being nacked by slave during master transfer.

[2.0.8]

- Bug Fixes
  - Fixed I2C_MasterSetBaudRate issue that MSTSCLLOW and MSTSCLHIGH are incorrect when MSTTIME is odd.

[2.0.7]

- Bug Fixes
  - Two dividers, CLKDIV and MSTTIME are used to configure baudrate. According to reference manual, in order to generate 400kHz baudrate, the clock frequency after CLKDIV must be less than 2mHz. Fixed the bug that, the clock frequency after CLKDIV may be larger than 2mHz using the previous calculation method.
  - Fixed MISRA 10.1 issues.
  - Fixed wrong baudrate calculation when feature FSL_FEATURE_I2C_PREPCLKFRG_8MHZ is enabled.

[2.0.6]

- New Features
  - Added master timeout self-recovery support for feature FSL_FEATURE_I2C_TIMEOUT_RECOVERY.
- Bug Fixes
  - Eliminated IAR Pa082 warning.
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 10.7, 10.8, 11.3, 11.6, 11.8, 11.9, 13.5.

[2.0.5]

- Bug Fixes
  - Fixed wrong assignment for datasize in I2C_InitTransferStateMachineDMA.
  - Fixed wrong working flow in I2C_RunTransferStateMachineDMA to ensure master can work in no start flag and no stop flag mode.
  - Fixed wrong working flow in I2C_RunTransferStateMachine and added kReceiveDataBeginState in _i2c_transfer_states to ensure master can work in no start flag and no stop flag mode.
  - Fixed wrong handle state in I2C_MasterTransferDMAHandleIRQ. After all the data has been transfered or nak is returned, handle state should be changed to idle.
- Improvements
  - Rounded up the calculated divider value in I2C_MasterSetBaudRate.

[2.0.4]

- Improvements
  - Updated the I2C_WATI_TIMEOUT macro to unified name I2C_RETRY_TIMES
  - Updated the "I2C_MasterSetBaudRate" API to support baudrate configuration for feature QN9090.
- Bug Fixes
  - Fixed build warnning caused by uninitialized variable.
  - Fixed COVERITY issue of unchecked return value in I2C_RTOS_Transfer.

[2.0.3]

- Improvements
  - Unified the component full name to FLEXCOMM I2C(DMA/FREERTOS) driver.

[2.0.2]

- Improvements
  - In slave IRQ:
    1. Changed slave receive process to first set the I2C_SLVCTL_SLVCONTINUE_MASK to acknowledge the received data, then do data receive.
    2. Improved slave transmit process to set the I2C_SLVCTL_SLVCONTINUE_MASK immediately after writing the data.

[2.0.1]

- Improvements
  - Added I2C_WATI_TIMEOUT macro to allow users to specify the timeout times for waiting flags in functional API and blocking transfer API.

[2.0.0]

- Initial version.

I2S

[2.3.2]

- Bug Fixes
  - Fixed warning for comparison between pointer and integer.

[2.3.1]

- Bug Fixes
  - Updated the value of TX/RX software transfer state machine after transfer contents are submitted to avoid race condition.

[2.3.0]

- Improvements
  - Added api I2S_InstallDMADescriptorMemory/I2S_TransferSendLoopDMA/I2S_TransferReceiveLoopD to support loop transfer.
  - Added api I2S_EmptyTxFifo to support blocking flush tx fifo.
  - Updated api I2S_TransferAbortDMA by removed the blocking flush tx fifo from this function.
- Bug Fixes
  - Removed the while loop in abort transfer function to fix the dead loop issue under specific user case.

[2.2.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4.

[2.2.1]

- Improvements
  - Added feature FSL_FEATURE_FLEXCOMM_INSTANCE_I2S_SUPPORT_SECONDARY_CHANNELn for the SOC has parts of instance support secondary channel.
- Bug Fixes
  - Added volatile statement for the state variable of i2s_handle and enable the mainline channel pair before enable interrupt to avoid the issue of code excution reordering which may cause the interrupt generated unexpectedly.

[2.2.0]

- Improvements
  - Added 8/16/24 bits mono data format transfer support in I2S driver.
  - Added new apis I2S_SetBitClockRate.
- Bug Fixes
  - Fixed the PA082 build warning.
  - Fixed the sign-compare warning.

  - Fixed violations of the MISRA C-2012 rules 10.4, 10.8, 11.9, 10.1, 11.3, 13.5, 11.8, 10.3, 10.7.
  - Fixed the Operand don't affect result Coverity issue.

[2.1.0]

- Improvements
  - Added a feature for the FLEXCOMM which supports I2S and has interconnection with DMIC.
  - Used a feature to control PDMDATA instead of I2S_CFG1_PDMDATA.
  - Added member bytesPerFrame in i2s_dma_handle_t, used for DMA transfer width configure, instead of using sizeof(uint32_t) hardcode.
  - Used the macro provided by DMA driver to define the I2S DMA descriptor.
- Bug Fixes
  - Fixed the issue that I2S DMA driver always generated duplicate callback.

[2.0.3]

- New Features
  - Added a feature to remove configuration for the second channel on LPC51U68.

[2.0.2]

- New Features
  - Added ENABLE_IRQ handle after register I2S interrupt handle.

[2.0.1]

- Improvements
  - Unified the component full name to FLEXCOMM I2S (DMA) driver.

[2.0.0]

- Initial version.

I2S_DMA

[2.3.3]

- Bug Fixes
  - Fixed data size limit does not match the macro DMA_MAX_TRANSFER_BYTES issue.

[2.3.2]

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

[2.3.1]

- Refer I2S driver change log 2.0.1 to 2.3.1

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_DMA

[2.1.11]

- Bug Fixes
  - Fixed the Coverity unused value issue in I3C_MasterSetTxDMA.

[2.1.10]

- Improvements
  - Added DMA chain optimization for Errata 052123.

[2.1.9]

- Improvements
  - Fixed Coverity CERT-C violations.

[2.1.8]

- Bug Fixes
  - Updated the logic to handle Rx termination and complete event to adapt different situation.
- Improvements
  - Added the MCTRLDONE flag check after STOP request to ensure the completion of whole transfer operation.

[2.1.7]

- Bug Fixes
  - Fixed the issue to use subaddress to read/write data with RT500/600 DMA.

[2.1.6]

- Improvements
  - Added the FSL_FEATURE_I3C_HAS_NO_MASTER_DMA_WDATA_REG to select the correct register to write data based on specific Soc.

[2.1.5]

- New features
  - Supported I3C HDR-DDR transfer with DMA.
- Improvements
  - Added workaround for RT500/600 I3C DMA transfer.
  - Removed I3C IRQ handler calling in the Tx EDMA callback. Previously driver doesn't use the END byte which can trigger the complete interrupt for controller sending and receiving, now let I3C event handler deal with I3C events.
  - Used linked DMA to transfer all I3C subaddress and data without handling of intermediate states, simplifying code logic.
  - Prepare the Tx DMA before I3C START to ensure there's no time delay between START and transmitting data.

[2.1.4]

- Improvements
  - Used linked DMA transfer to reduce the latency between DMA transfers previous data and the END byte.

[2.1.3]

- Bug Fixes
  - Fixed the MISRA issue rule 10.4, 11.3.

[2.1.2]

- Bug Fixes
  - Fixed the issue that I3C slave send the last byte data without using the END type register.

[2.1.1]

- Bug Fixes
  - Fixed MISRA issue rule 9.1.

[2.1.0]

- Improvements
  - Deleted legacy IBI data request code.

[2.0.1]

- Bug Fixes
  - Fixed issue that bus STOP occurs when Tx FIFO still takes data.
- Improvements
  - Fixed the build warning due to armgcc strict check.

[2.0.0]

- Initial version.

IAP

[2.1.3]

- Bug Fixes
  - Fixed misra issue.

[2.1.2]

- Bug Fixes
  - Fixed some macro undefined issue.
  - Put IAP_FlexspiNorInit API into RAM.

[2.1.1]

- Bug Fixes
  - Fixed misra issue.

[2.1.0]

- New Features
  - Added IAP_RunBootLoader() API

[2.0.2]

- Bug Fixes
  - Fixed doxygen issue.

[2.0.1]

- Bug Fixes
  - Minor update for MISRA issue fix.

[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

[2.0.0]

- 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.

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

[2.3.3]

- Improvements
  - Fix build failure when driver clock control disabled.

[2.3.2]

- Improvements
  - Fix MISRA C-2012 Rule 20.8 violations.

[2.3.1]

- Bug Fixes
  - Fixed FSL_FEATURE_MU_HAS_RESET_DEASSERT_INT macro use.

[2.3.0]

- New Features
  - Added MU_BUSY_POLL_COUNT parameter to prevent infinite polling loops in MU operations.
  - Added timeout mechanism to all polling loops in MU driver code.
  - Added new function MU_ReceiveMsgTimeout() to include timeout mechanism.
- 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
    - Updated MU_ReceiveMsg to use MU_TIMEOUT_VALUE (0xFFFFFFFF) as a special return value to indicate timeout
  - Enhanced documentation across all functions to clarify timeout behavior and return values.

[2.2.0]

- New Features
  - Added API MU_GetRxStatusFlags.

[2.1.3]

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

[2.1.2]

- Bug Fixes
  - Fixed issue that MU_GetInstance() is defined but never used.

[2.1.1]

- Bug Fixes
  - Fixed general interrupt comment typo.

[2.1.0]

- Improvements
  - Added new enum mu_msg_reg_index_t.

[2.0.7]

- Bug Fixes
  - Fixed MU_GetInterruptsPending bug that can not get general interrupt status.

[2.0.6]

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

[2.0.5]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 14.4, 15.5.

[2.0.4]

- Improvements
  - Improved for the platforms which don't support reset assert interrupt and get the other core power mode.

[2.0.3]

- Bug fixes
  - MISRA C-2012 issue fixed.
    - Fixed rules, containing: rule-10.3, rule-14.4, rule-15.5.

[2.0.2]

- Improvements
  - Added support for MIMX8MQx.

[2.0.1]

- Improvements
  - Added support for MCIMX7Ux_M4.

[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.

OTFAD

[2.1.4]

- Bug fixes
  - Fixed MISRA 2012 issue: 10.1.
  - Fixed CERT INT31-C violations: bool fields cast to unsigned long in register macros now use explicit ternary.
  - Fixed CERT INT30-C violation: guarded totalHits increment in OTFAD_HitDetermination against overflow.

[2.1.3]

- Bug fixes
  - Fixed the error that waiting for both FLEXSPI AHB idle and SEQ idle.

[2.1.2]

- Bug fixes
  - Fixed MISRA 2012 issue: 10.4.

[2.1.1]

- Improvements:
  - Hided some bits in CR and SR registers for selected platforms.
  - Fixed doxygen issues.

[2.1.0]

- Improvements:
  - Used boolean type to define 1-bit field concepts.

[2.0.0]

- Initial version.

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.

POWER

[2.5.1]

- Improvement
  - Support customizing default PMIC mode for low power modes by overriding macro value of POWER_DEFAULT_PMICMODE_xx.

[2.5.0]

- New feature
  - Added new API POWER_PmicPowerModeSelectControl() to allow users changing VDD1V8 and VDDCore state for various PMIC modes.

[2.4.0]

- New feature
  - Added new API POWER_SetVddCoreSupplySrc(), POWER_SetPmicCoreSupplyFunc() and POWER_SetVoltageForFreq() to allow users set VDDCORE voltage using a unified API with minimum volatage value.

[2.3.2]

- Bug Fixes
  - Fixed MISRA issue in function POWER_GetLibVersion.

[2.3.1]

- Bug Fixes
  - Fixed the return value of function countPartitionSwitches.

[2.3.0]

- API change: POWER_SetLdoVoltageForFreq()
  - Changed power_part_temp_range_t. 70C->85C.
  - Added parameter power_volt_op_range_t.
  - Changed main clock freq parameter to cpu clock freq.

  - Added return value to indicate success or failure.
- Optimization
  - Turn on all memory partitions simultaneously on deep sleep wakeup to save time.
- Release in source code instead of in library.

[2.2.1]

- Exposed POWER_DisableLVD() and POWER_RestoreLVD() APIs in header.
- Added PMIC_VDDCORE_RECOVERY_TIME_IGNORE macro for POWER_UpdatePmicRecoveryTime() API.
- Adjusted main frequency table for SetXXXVoltageForFreq() API to match latest B0 data.

[2.2.0]

- Added parameter to POWER_SetLdoVoltageForFreq to specify part temperature range.

[2.1.0]

- Updated power library implementation for B0.
- Added POWER_SetLvdFallingTripVoltage() API.
- Added POWER_GetLvdFallingTripVoltage() API.
- Added POWER_UpdatePmicRecoveryTime() API.

[2.0.3]

- Updated PD_bits per the B0 header file's change.

[2.0.2]

- Added POWER_SetPadVolRange() API

[2.0.1]

- Add POWER_UpdateOscSettlingTime() API to set on-board system osc settling time.

[2.0.0]

- initial version.

POWERQUAD

[2.2.1]

- Bug Fixes
  - Fixed CERT-C issues.

[2.2.0]

- New Features
  - Added new API PQ_Arctan2Fixed.

[2.1.1]

- Bug Fixes
  - Remove PQ_WaitDone from PQ_ArctanFixed and PQ_ArctanhFixed because it is unnecessary.

[2.1.0]

- Improvements
  - Fixed typo issue for biquad related function name.
  - Changed operator from "%" into "&" to reduce heavy cycle for biquad functions.

[2.0.5]

- Improvements
  - Added a note in driver for FIR that powerquad has a hardware limitation, when using it for FIR increment calculation, the address of pSrc needs to be a continuous address.

[2.0.4]

- Improvements
  - Supported the platforms which don't have PowerQuad clock and reset control.

[2.0.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 8.4, 10.1, 10.3, 10.4, 10.6, and so on.

[2.0.2]

- Bug Fixes
  - Fixed array size issue in fsl_powerquad_data.h file.
  - Fixed vector function pipeline issue.

[2.0.1]

- Bug Fixes
  - Fixed build error in C++ mode.

[2.0.0]

- Initial version.

POWERQUAD_CMSIS

[2.1.0]

- Improvements
  - Support use different API name, to work together with CMSIS DSP software implementation.

[2.0.4]

- Inherit the SDK 2.0 driver.

PUF

[2.2.1]

- Fix CERT-C violations:
  - ARR30-C / OVERRUN in PUF_GetHwKey(): bound the KEYMASK[] register index by PUF_KEYMASK_COUNT to avoid out-of-bounds access.
  - INT30-C in PUF_GetKey(): add keySize range check to prevent unsigned integer wraparound.

[2.2.0]

- Add support for kPUF_KeySlot4.
- Add new PUF_ClearKey() function, that clears a desired PUF internal HW key register.

[2.1.6]

- Changed wait time in PUF_Init(), when initialization fails it will try PUF_Powercycle() with shorter time. If this shorter time will also fail, initialization will be tried with worst case time as before.

[2.1.5]

- Use common SDK delay in puf_wait_usec().

[2.1.4]

- Replace register uint32_t ticksCount with volatile uint32_t ticksCount in puf_wait_usec() to prevent optimization out delay loop.

[2.1.3]

- Fix MISRA C-2012 issue.

[2.1.2]

- Update: Add automatic big to little endian swap for user (pre-shared) keys destinated to secret hardware bus (PUF key index 0).

[2.1.1]

- Fix ARMGCC build warning .

[2.1.0]

- Align driver with PUF SRAM controller registers on LPCXpresso55s16.
- Update initizalition logic .

[2.0.3]

- Fix MISRA C-2012 issue.

[2.0.2]

- New feature:
  - Add PUF configuration structure and support for PUF SRAM controller.
- Improvements:
  - Remove magic constants.

[2.0.1]

- Bug Fixes:
  - Fixed puf_wait_usec function optimization issue.

[2.0.0]

- Initial version.

RESET

[2.2.2]

- Improvements
  - Added reset array for several peripherals

[2.2.1]

- Bug Fiexs
  - Added peripheral reset array for MU.

[2.2.0]

- Improvements
  - Add RESET_ReleasePeripheralReset API.

[2.1.3]

- Bug Fixes
  - Added peripheral reset array for CASPER, PUF, HashCrypt, RNG.

[2.1.2]

- Bug Fixes
  - Fixed typo in _RSTCTL_RSTn enumeration's comment.

[2.1.1]

- Bug Fixes
  - Fixed MISRA C-2012 rule 10.6 and rule 16.4.

[2.1.0]

- Updated register access per the B0 header file's change.

[2.0.4]

- Add SDIO1 and Flexcomm6,7 support.

[2.0.3]

- Rename RSTCTRL to RSTCTL.

[2.0.2]

- Update according to UM 0.7.

[2.0.1]

- Update component full_name to "Reset Driver".

[2.0.0]

- initial version.

RTC

[2.2.0]

- New Features
  - Created new APIs for the RTC driver.
    - RTC_EnableSubsecCounter
    - RTC_GetSubsecValue

[2.1.3]

- Bug Fixes
  - Fixed issue that RTC_GetWakeupCount may return wrong value.

[2.1.2]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.1, 10.4 and 10.7.

[2.1.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.3 and 11.9.

[2.1.0]

- Bug Fixes
  - Created new APIs for the RTC driver.
    - RTC_EnableTimer
    - RTC_EnableWakeUpTimerInterruptFromDPD
    - RTC_EnableAlarmTimerInterruptFromDPD
    - RTC_EnableWakeupTimer
    - RTC_GetEnabledWakeupTimer
    - RTC_SetSecondsTimerMatch
    - RTC_GetSecondsTimerMatch
    - RTC_SetSecondsTimerCount
    - RTC_GetSecondsTimerCount
  - deprecated legacy APIs for the RTC driver.
    - RTC_StartTimer
    - RTC_StopTimer
    - RTC_EnableInterrupts
    - RTC_DisableInterrupts
    - RTC_GetEnabledInterrupts

[2.0.0]

- Initial version.

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.

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.

SPI

[2.4.0]

- New Features
  - Added kSPI_FrameAssertEachFrame to deassert PCS at the end of every frame.

[2.3.2]

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

[2.3.1]

- Improvements
  - Changed SPI_DUMMYDATA to 0x00.

[2.3.0]

- Update version.

[2.2.2]

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

[2.2.1]

- Bug Fixes
  - Fixed MISRA 2012 10.4 issue.
  - Added code to clear FIFOs before transfer using DMA.

[2.2.0]

- Bug Fixes
  - Fixed bug that slave gets stuck during interrupt transfer.

[2.1.1]

- Improvements
  - Added timeout mechanism when waiting certain states in transfer driver.
- Bug Fixes
  - Fixed MISRA 10.1, 5.7 issues.

[2.1.0]

- Bug Fixes
  - Fixed Coverity issue of incrementing null pointer in SPI_TransferHandleIRQInternal.
  - Eliminated IAR Pa082 warnings.
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 10.7, 10.8, 11.3, 11.6, 11.8, 11.9, 13.5.
- New Features
  - Modified the definition of SPI_SSELPOL_MASK to support the socs that have only 3 SSEL pins.

[2.0.4]

- Bug Fixes
  - Fixed the bug of using read only mode in DMA transfer. In DMA transfer mode, if transfer->txData is NULL, code attempts to read data from the address of 0x0 for configuring the last frame.
  - Fixed wrong assignment of handle->state. During transfer handle->state should be kSPI_Busy rather than kStatus_SPI_Busy.
- Improvements
  - Rounded up the calculated divider value in SPI_MasterSetBaud.

[2.0.3]

- Improvements
  - Added "SPI_FIFO_DEPTH(base)" with more definition.

[2.0.2]

- Improvements
  - Unified the component full name to FLEXCOMM SPI(DMA/FREERTOS) driver.

[2.0.1]

- Changed the data buffer from uint32_t to uint8_t which matches the real applications for SPI DMA driver.
- Added dummy data setup API to allow users to configure the dummy data to be transferred.
- Added new APIs for half-duplex transfer function. Users can not only send and receive data by one API in polling/interrupt/DMA way, but choose either to transmit first or to receive first. Besides, the PCS pin can be configured as assert status in transmission (between transmit and receive) by setting the isPcsAssertInTransfer to true.

[2.0.0]

- Initial version.

SPI_DMA

[2.3.0]

- Improvements
  - Added support for kSPI_FrameAssertEachFrame so DMA transfers can deassert PCS after each frame.

[2.2.3]

- Bug Fixes
  - Fixed the bug SPI_MasterTransferGetCountDMA and SPI_SlaveTransferGetCountDMA returns wrong count.

[2.2.2]

- Bug Fixes
  - Fixed the bug half duplex mode can't be used if data size is larger than 1024 bytes.

[2.2.1]

- Bug Fixes
  - Fixed MISRA 2012 11.6 issue..

[2.2.0]

- Improvements
  - Supported dataSize larger than 1024 data transmit.

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.

USART

[2.9.0]

- New Features
  - Added support for RS-485 (configuration of OESEL, OEPOL, OETA in CFG register).
  - Added handling when a start is detected on the receiver input in USART_TransferHandleIRQ().

[2.8.5]

- Bug Fixes
  - Fixed race condition during call of USART_EnableTxDMA and USART_EnableRxDMA.

[2.8.4]

- Bug Fixes
  - Fixed exclusive access in USART_TransferReceiveNonBlocking and USART_TransferSendNonBlocking.

[2.8.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 11.8.

[2.8.2]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 14.2.

[2.8.1]

- Bug Fixes
  - Fixed the Baud Rate Generator(BRG) configuration in 32kHz mode.

[2.8.0]

- New Features
  - Added the rx timeout interrupts and status flags of bus status.
  - Added new rx timeout configuration item in usart_config_t.
  - Added API USART_SetRxTimeoutConfig for rx timeout configuration.
- Improvements
  - When the calculated baudrate cannot meet user's configuration, lower OSR value is allewed to use.

[2.7.0]

- New Features
  - Added the missing interrupts and status flags of bus status.
  - Added the check of tx error, noise error framing error and parity error in interrupt handler.

[2.6.0]

- Improvements
  - Used separate data for TX and RX in usart_transfer_t.
- Bug Fixes
  - Fixed bug that when ring buffer is used, if some data is received in ring buffer first before calling USART_TransferReceiveNonBlocking, the received data count returned by USART_TransferGetReceiveCount is wrong.
- New Features
  - Added missing API USART_TransferGetSendCountDMA get send count using DMA.

[2.5.0]

- New Features
  - Added APIs USART_GetRxFifoCount/USART_GetTxFifoCount to get rx/tx FIFO data count.
  - Added APIs USART_SetRxFifoWatermark/USART_SetTxFifoWatermark to set rx/tx FIFO water mark.
- Bug Fixes
  - Fixed DMA transfer blocking issue by enabling tx idle interrupt after DMA transmission finishes.

[2.4.0]

- New Features
  - Modified usart_config_t, USART_Init and USART_GetDefaultConfig APIs so that the hardware flow control can be enabled during module initialization.
- Bug Fixes
  - Fixed MISRA 10.4 violation.

[2.3.1]

- Bug Fixes
  - Fixed bug that operation on INTENSET, INTENCLR, FIFOINTENSET and FIFOINTENCLR should use bitwise operation not ‘or’ operation.
  - Fixed bug that if rx interrupt occurrs before TX interrupt is enabled and after txDataSize is configured, the data will be sent early by mistake, thus TX interrupt will be enabled after data is sent out.
- Improvements

  - Added check for baud rate's accuracy that returns kStatus_USART_BaudrateNotSupport when the best achieved baud rate is not within 3% error of configured baud rate.

[2.3.0]

- New Features
  - Added APIs to configure 9-bit data mode, set slave address and send address.
  - Modified USART_TransferReceiveNonBlocking and USART_TransferHandleIRQ to use 9-bit mode in multi-slave system.

[2.2.0]

- New Features
  - Added the feature of supporting USART working at 32 kHz clocking mode.
- Improvements
  - Modified USART_TransferHandleIRQ so that txState will be set to idle only when all data has been sent out to bus.
  - Modified USART_TransferGetSendCount so that this API returns the real byte count that USART has sent out rather than the software buffer status.
  - Added timeout mechanism when waiting for certain states in transfer driver.
- Bug Fixes
  - Fixed MISRA 10.1 issues.
  - Fixed bug that operation on INTENSET, INTENCLR, FIFOINTENSET and FIFOINTENCLR should use bitwise operation not 'or' operation.
  - Fixed bug that if rx interrupt occurrs before TX interrupt is enabled and after txDataSize is configured, the data will be sent early by mistake, thus TX interrupt will be enabled after data is sent out.

[2.1.1]

- Improvements
  - Added check for transmitter idle in USART_TransferHandleIRQ and USART_TransferSendDMACallback to ensure all the data would be sent out to bus.
  - Modified USART_ReadBlocking so that if more than one receiver errors occur, all status flags will be cleared and the most severe error status will be returned.
- Bug Fixes
  - Eliminated IAR Pa082 warnings.
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 10.7, 10.8, 11.3, 11.6, 11.8, 11.9, 13.5.

[2.1.0]

- New Features
  - Added features to allow users to configure the USART to synchronous transfer(master and slave) mode.
- Bug Fixes

  - Modified USART_SetBaudRate to get more acurate configuration.

[2.0.3]

- New Features
  - Added new APIs to allow users to enable the CTS which determines whether CTS is used for flow control.

[2.0.2]

- Bug Fixes
  - Fixed the bug where transfer abort APIs could not disable the interrupts. The FIFOINTENSET register should not be used to disable the interrupts, so use the FIFOINTENCLR register instead.

[2.0.1]

- Improvements
  - Unified the component full name to FLEXCOMM USART (DMA/FREERTOS) driver.

[2.0.0]

- Initial version.

USART_DMA

[2.6.0]

- Refer USART driver change log 2.0.1 to 2.6.0

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.

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.

MIMXRT685S

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 Multicore

Multicore SDK

1.9.2 MCU Boot

mcuboot_opensource

1.9.3 eIQ

eiq

1.9.4 FreeMASTER

freemaster

1.9.5 NXP Wi-Fi

Wi-Fi, Bluetooth, 802.15.4

1.9.6 FreeRTOS

FreeRTOS

1.9.7 lwIP

lwIP

1.9.8 File systemFatfs

FatFs

1.9.9 DSP Audio Streamer

Xtensa Audio Framework (XAF)

Chapter 2

MIMXRT685S

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 CASPER: The Cryptographic Accelerator and Signal Processing Engine with RAM sharing

2.4 casper_driver

FSL_CASPER_DRIVER_VERSION
CASPER driver version. Version 2.2.5.

Current version: 2.2.5

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Bug fix KPSDK-24531 double_scalar_multiplication() result may be all zeroes for some specific input
- Version 2.0.2
  - Bug fix KPSDK-25015 CASPER_MEMCPY hard-fault on LPC55xx when both source and destination buffers are outside of CASPER_RAM
- Version 2.0.3
  - Bug fix KPSDK-28107 RSUB, FILL and ZERO operations not implemented in enum _casper_operation.
- Version 2.0.4
  - For GCC compiler, enforce O1 optimize level, specifically to remove strict-aliasing option. This driver is very specific and requires -fno-strict-aliasing.

- Version 2.0.5
  - Fix sign-compare warning.
- Version 2.0.6
  - Fix IAR Pa082 warning.
- Version 2.0.7
  - Fix MISRA-C 2012 issue.
- Version 2.0.8
  - Add feature macro for CASPER_RAM_OFFSET.
- Version 2.0.9
  - Remove unused function Jac_oncurve().
  - Fix ECC384 build.
- Version 2.0.10
  - Fix MISRA-C 2012 issue.
- Version 2.1.0
  - Add ECC NIST P-521 elliptic curve.
- Version 2.2.0
  - Rework driver to support multiple curves at once.
- Version 2.2.1
  - Fix MISRA-C 2012 issue.
- Version 2.2.2
  - Enable hardware interleaving to RAMX0 and RAMX1 for CASPER by feature macro FSL_FEATURE_CASPER_RAM_HW_INTERLEAVE
- Version 2.2.3
  - Added macro into CASPER_Init and CASPER_Deinit to support devices without clock and reset control.
- Version 2.2.4
  - Fix MISRA-C 2012 issue.
- Version 2.2.5
  - Fix Coverity Uninitialized scalar variable.

enum _casper_operation

CASPER operation.

Values:

enumerator kCASPER_OpMul6464NoSum

enumerator kCASPER_OpMul6464Sum

Walking 1 or more of J loop, doing r=a*b using 64x64=128

enumerator kCASPER_OpMul6464FullSum

Walking 1 or more of J loop, doing c,r=r+a*b using 64x64=128, but assume inner j loop

enumerator kCASPER_OpMul6464Reduce

Walking 1 or more of J loop, doing c,r=r+a*b using 64x64=128, but sum all of w.

enumerator kCASPER_OpAdd64

Walking 1 or more of J loop, doing c,r[-1]=r+a*b using 64x64=128, but skip 1st write

enumerator kCASPER_OpSub64

Walking add with off_AB, and in/out off_RES doing c,r=r+a+c using 64+64=65

enumerator kCASPER_OpDouble64

Walking subtract with off_AB, and in/out off_RES doing r=r-a using 64-64=64, with last borrow implicit if any

enumerator kCASPER_OpXor64

Walking add to self with off_RES doing c,r=r+r+c using 64+64=65

enumerator kCASPER_OpRSub64

Walking XOR with off_AB, and in/out off_RES doing r=r^a using 64^64=64

enumerator kCASPER_OpShiftLeft32

Walking subtract with off_AB, and in/out off_RES using r=a-r

enumerator kCASPER_OpShiftRight32

Walking shift left doing r1,r=(b*D)|r1, where D is 2^amt and is loaded by app (off_CD not used)

enumerator kCASPER_OpCopy

Walking shift right doing r,r1=(b*D)|r1, where D is 2^(32-amt) and is loaded by app (off_CD not used) and off_RES starts at MSW

enumerator kCASPER_OpRemask

Copy from ABoff to resoff, 64b at a time

enumerator kCASPER_OpFill

Copy and mask from ABoff to resoff, 64b at a time

enumerator kCASPER_OpZero

Fill RESOFF using 64 bits at a time with value in A and B

enumerator kCASPER_OpCompare

Fill RESOFF using 64 bits at a time of 0s

enumerator kCASPER_OpCompareFast

Compare two arrays, running all the way to the end

enum _casper_algo_t

Algorithm used for CASPER operation.

Values:

enumerator kCASPER_ECC_P256
ECC_P256

enumerator kCASPER_ECC_P384
ECC_P384

enumerator kCASPER_ECC_P521
ECC_P521

Values:

enumerator kCASPER_RamOffset_Result

enumerator kCASPER_RamOffset_Base

enumerator kCASPER_RamOffset_TempBase

enumerator kCASPER_RamOffset_Modulus

enumerator kCASPER_RamOffset_M64

typedef enum *_casper_operation* casper_operation_t
CASPER operation.

typedef enum *_casper_algo_t* casper_algo_t
Algorithm used for CASPER operation.

void CASPER_Init(CASPER_Type *base)
Enables clock and disables reset for CASPER peripheral.

Enable clock and disable reset for CASPER.

Parameters

- base – CASPER base address

void CASPER_Deinit(CASPER_Type *base)
Disables clock for CASPER peripheral.

Disable clock and enable reset.

Parameters

- base – CASPER base address

CASPER_CP

CASPER_CP_CTRL0

CASPER_CP_CTRL1

CASPER_CP_LOADER

CASPER_CP_STATUS

CASPER_CP_INTENSET

CASPER_CP_INTENCLR

CASPER_CP_INTSTAT

CASPER_CP_AREG

CASPER_CP_BREG

CASPER_CP_CREG

CASPER_CP_DREG

CASPER_CP_RES0

CASPER_CP_RES1

CASPER_CP_RES2

CASPER_CP_RES3

CASPER_CP_MASK

CASPER_CP_REMASK

CASPER_CP_LOCK

CASPER_CP_ID

CASPER_Wr32b(value, off)

CASPER_Wr64b(value, off)

CASPER_Rd32b(off)

N_wordlen_max

2.5 casper_driver_pkha

void CASPER_ModExp(CASPER_Type *base, const uint8_t *signature, const uint8_t *pubN, size_t wordLen, uint32_t pubE, uint8_t *plaintext)

Performs modular exponentiation - (A^E) mod N.

This function performs modular exponentiation.

Parameters

- base – CASPER base address
- signature – first addend (in little endian format)
- pubN – modulus (in little endian format)
- wordLen – Size of pubN in bytes
- pubE – exponent
- plaintext – **[out]** Output array to store result of operation (in little endian format)

void CASPER_ecc_init(*casper_algo_t* curve)

Initialize prime modulus mod in Casper memory .

Set the prime modulus mod in Casper memory and set N_wordlen according to selected algorithm.

Parameters

- curve – elliptic curve algoritm

void CASPER_ECC_SECP256R1_Mul(CASPER_Type *base, uint32_t resX[8], uint32_t resY[8], uint32_t X[8], uint32_t Y[8], uint32_t scalar[8])

Performs ECC secp256r1 point single scalar multiplication.

This function performs ECC secp256r1 point single scalar multiplication [resX; resY] = scalar * [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate in normal form, little endian.
- resY – **[out]** Output Y affine coordinate in normal form, little endian.
- X – Input X affine coordinate in normal form, little endian.
- Y – Input Y affine coordinate in normal form, little endian.
- scalar – Input scalar integer, in normal form, little endian.

void CASPER_ECC_SECP256R1_MulAdd(CASPER_Type *base, uint32_t resX[8], uint32_t resY[8], uint32_t X1[8], uint32_t Y1[8], uint32_t scalar1[8], uint32_t X2[8], uint32_t Y2[8], uint32_t scalar2[8])

Performs ECC secp256r1 point double scalar multiplication.

This function performs ECC secp256r1 point double scalar multiplication [resX; resY] = scalar1 * [X1; Y1] + scalar2 * [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate.
- resY – **[out]** Output Y affine coordinate.
- X1 – Input X1 affine coordinate.
- Y1 – Input Y1 affine coordinate.

- scalar1 – Input scalar1 integer.
- X2 – Input X2 affine coordinate.
- Y2 – Input Y2 affine coordinate.
- scalar2 – Input scalar2 integer.

void CASPER_ECC_SECP384R1_Mul(CASPER_Type *base, uint32_t resX[12], uint32_t resY[12], uint32_t X[12], uint32_t Y[12], uint32_t scalar[12])

Performs ECC secp384r1 point single scalar multiplication.

This function performs ECC secp384r1 point single scalar multiplication [resX; resY] = scalar * [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate in normal form, little endian.
- resY – **[out]** Output Y affine coordinate in normal form, little endian.
- X – Input X affine coordinate in normal form, little endian.
- Y – Input Y affine coordinate in normal form, little endian.
- scalar – Input scalar integer, in normal form, little endian.

void CASPER_ECC_SECP384R1_MulAdd(CASPER_Type *base, uint32_t resX[12], uint32_t resY[12], uint32_t X1[12], uint32_t Y1[12], uint32_t scalar1[12], uint32_t X2[12], uint32_t Y2[12], uint32_t scalar2[12])

Performs ECC secp384r1 point double scalar multiplication.

This function performs ECC secp384r1 point double scalar multiplication [resX; resY] = scalar1 * [X1; Y1] + scalar2 * [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate.
- resY – **[out]** Output Y affine coordinate.
- X1 – Input X1 affine coordinate.
- Y1 – Input Y1 affine coordinate.
- scalar1 – Input scalar1 integer.
- X2 – Input X2 affine coordinate.
- Y2 – Input Y2 affine coordinate.
- scalar2 – Input scalar2 integer.

void CASPER_ECC_SECP521R1_Mul(CASPER_Type *base, uint32_t resX[18], uint32_t resY[18], uint32_t X[18], uint32_t Y[18], uint32_t scalar[18])

Performs ECC secp521r1 point single scalar multiplication.

This function performs ECC secp521r1 point single scalar multiplication [resX; resY] = scalar * [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate in normal form, little endian.
- resY – **[out]** Output Y affine coordinate in normal form, little endian.
- X – Input X affine coordinate in normal form, little endian.
- Y – Input Y affine coordinate in normal form, little endian.
- scalar – Input scalar integer, in normal form, little endian.

void CASPER_ECC_SECP521R1_MulAdd(CASPER_Type *base, uint32_t resX[18], uint32_t resY[18], uint32_t X1[18], uint32_t Y1[18], uint32_t scalar1[18], uint32_t X2[18], uint32_t Y2[18], uint32_t scalar2[18])

Performs ECC secp521r1 point double scalar multiplication.

This function performs ECC secp521r1 point double scalar multiplication [resX; resY] = scalar1 * [X1; Y1] + scalar2 * [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

- base – CASPER base address
- resX – **[out]** Output X affine coordinate.
- resY – **[out]** Output Y affine coordinate.
- X1 – Input X1 affine coordinate.
- Y1 – Input Y1 affine coordinate.
- scalar1 – Input scalar1 integer.
- X2 – Input X2 affine coordinate.
- Y2 – Input Y2 affine coordinate.
- scalar2 – Input scalar2 integer.

void CASPER_ECC_equal(int *res, uint32_t *op1, uint32_t *op2)

void CASPER_ECC_equal_to_zero(int *res, uint32_t *op1)

2.6 Clock Driver

enum _clock_ip_name

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

Values:

enumerator kCLOCK_IpInvalid

Invalid Ip Name.

enumerator kCLOCK_RomCtrlr

Clock gate name: RomCtrlr

enumerator kCLOCK_PowerQuad
Clock gate name: PowerQuad

enumerator kCLOCK_Casper
Clock gate name: Casper

enumerator kCLOCK_HashCrypt
Clock gate name: HashCrypt

enumerator kCLOCK_Puf
Clock gate name: Puf

enumerator kCLOCK_Rng
Clock gate name: Rng

enumerator kCLOCK_Flexspi
Clock gate name: Flexspi

enumerator kCLOCK_OtpCtrl
Clock gate name: OtpCtrl

enumerator kCLOCK_UsbhsPhy
Clock gate name: UsbhsPhy

enumerator kCLOCK_UsbhsDevice
Clock gate name: UsbhsDevice

enumerator kCLOCK_UsbhsHost
Clock gate name: UsbhsHost

enumerator kCLOCK_UsbhsSram
Clock gate name: UsbhsSram

enumerator kCLOCK_Sct
Clock gate name: Sct

enumerator kCLOCK_Sdio0
Clock gate name: Sdio0

enumerator kCLOCK_Sdio1
Clock gate name: Sdio1

enumerator kCLOCK_Acmp0
Clock gate name: Acmp0

enumerator kCLOCK_Adc0
Clock gate name: Adc0

enumerator kCLOCK_ShsGpio0
Clock gate name: ShsGpio0

enumerator kCLOCK_Utick0
Clock gate name: Utick0

enumerator kCLOCK_Wwdt0
Clock gate name: Wwdt0

enumerator kCLOCK_Flexcomm0
Clock gate name: Flexcomm0

enumerator kCLOCK_Flexcomm1
Clock gate name: Flexcomm1

enumerator kCLOCK_Flexcomm2
Clock gate name: Flexcomm2

enumerator kCLOCK_Flexcomm3
Clock gate name: Flexcomm3

enumerator kCLOCK_Flexcomm4
Clock gate name: Flexcomm4

enumerator kCLOCK_Flexcomm5
Clock gate name: Flexcomm5

enumerator kCLOCK_Flexcomm6
Clock gate name: Flexcomm6

enumerator kCLOCK_Flexcomm7
Clock gate name: Flexcomm7

enumerator kCLOCK_Flexcomm14
Clock gate name: Flexcomm14

enumerator kCLOCK_Flexcomm15
Clock gate name: Flexcomm15

enumerator kCLOCK_Usart0
Clock gate name: Usart0

enumerator kCLOCK_Usart1
Clock gate name: Usart1

enumerator kCLOCK_Usart2
Clock gate name: Usart2

enumerator kCLOCK_Usart3
Clock gate name: Usart3

enumerator kCLOCK_Usart4
Clock gate name: Usart4

enumerator kCLOCK_Usart5
Clock gate name: Usart5

enumerator kCLOCK_Usart6
Clock gate name: Usart6

enumerator kCLOCK_Usart7
Clock gate name: Usart7

enumerator kCLOCK_I2s0
Clock gate name: I2s0

enumerator kCLOCK_I2s1
Clock gate name: I2s1

enumerator kCLOCK_I2s2
Clock gate name: I2s2

enumerator kCLOCK_I2s3
Clock gate name: I2s3

enumerator kCLOCK_I2s4
Clock gate name: I2s4

enumerator kCLOCK_I2s5
Clock gate name: I2s5

enumerator kCLOCK_I2s6
Clock gate name: I2s6

enumerator kCLOCK_I2s7
Clock gate name: I2s7

enumerator kCLOCK_I2c0
Clock gate name: I2c0

enumerator kCLOCK_I2c1
Clock gate name: I2c1

enumerator kCLOCK_I2c2
Clock gate name: I2c2

enumerator kCLOCK_I2c3
Clock gate name: I2c3

enumerator kCLOCK_I2c4
Clock gate name: I2c4

enumerator kCLOCK_I2c5
Clock gate name: I2c5

enumerator kCLOCK_I2c6
Clock gate name: I2c6

enumerator kCLOCK_I2c7
Clock gate name: I2c7

enumerator kCLOCK_I2c15
Clock gate name: I2c15

enumerator kCLOCK_Spi0
Clock gate name: Spi0

enumerator kCLOCK_Spi1
Clock gate name: Spi1

enumerator kCLOCK_Spi2
Clock gate name: Spi2

enumerator kCLOCK_Spi3
Clock gate name: Spi3

enumerator kCLOCK_Spi4
Clock gate name: Spi4

enumerator kCLOCK_Spi5
Clock gate name: Spi5

enumerator kCLOCK_Spi6
Clock gate name: Spi6

enumerator kCLOCK_Spi7
Clock gate name: Spi7

enumerator kCLOCK_Spi14
Clock gate name: Spi14

enumerator kCLOCK_Dmic0
Clock gate name: Dmic0

enumerator kCLOCK_OsEventTimer
Clock gate name: OsEventTimer

enumerator kCLOCK_HsGpio0
Clock gate name: HsGpio0

enumerator kCLOCK_HsGpio1
Clock gate name: HsGpio1

enumerator kCLOCK_HsGpio2
Clock gate name: HsGpio2

enumerator kCLOCK_HsGpio3
Clock gate name: HsGpio3

enumerator kCLOCK_HsGpio4
Clock gate name: HsGpio4

enumerator kCLOCK_HsGpio5
Clock gate name: HsGpio5

enumerator kCLOCK_HsGpio6
Clock gate name: HsGpio6

enumerator kCLOCK_HsGpio7
Clock gate name: HsGpio7

enumerator kCLOCK_Crc
Clock gate name: Crc

enumerator kCLOCK_Dmac0
Clock gate name: Dmac0

enumerator kCLOCK_Dmac1
Clock gate name: Dmac1

enumerator kCLOCK_Mu
Clock gate name: Mu

enumerator kCLOCK_Sema
Clock gate name: Sema

enumerator kCLOCK_Freqme
Clock gate name: Freqme

enumerator kCLOCK_Ct32b0
Clock gate name: Ct32b0

enumerator kCLOCK_Ct32b1
Clock gate name: Ct32b1

enumerator kCLOCK_Ct32b2
Clock gate name: Ct32b2

enumerator kCLOCK_Ct32b3
Clock gate name: Ct32b3

enumerator kCLOCK_Ct32b4
Clock gate name: Ct32b4

enumerator kCLOCK_Rtc
Clock gate name: Rtc

enumerator kCLOCK_Mrt0
Clock gate name: Mrt0

enumerator kCLOCK_Wwdt1
Clock gate name: Wwdt1

enumerator kCLOCK_I3c0
Clock gate name: I3c0

enumerator kCLOCK_Pint
Clock gate name: Pint

enumerator kCLOCK_InputMux
Clock gate name: Input Mux.

enum _clock_name
Clock name used to get clock frequency.
Values:

enumerator kCLOCK_CoreSysClk
Core clock (aka HCLK)

enumerator kCLOCK_BusClk
Bus clock (AHB/APB clock, aka HCLK)

enumerator kCLOCK_MclkClk
MCLK, to MCLK pin

enumerator kCLOCK_ClockOutClk
CLOCKOUT

enumerator kCLOCK_AdcClk
ADC

enumerator kCLOCK_FlexspiClk
FLEXSPI

enumerator kCLOCK_SctClk
SCT

enumerator kCLOCK_Wdt0Clk
Watchdog0

enumerator kCLOCK_Wdt1Clk
Watchdog1

enumerator kCLOCK_SystickClk
Systick

enumerator kCLOCK_Sdio0Clk
SDIO0

enumerator kCLOCK_Sdio1Clk
SDIO1

enumerator kCLOCK_I3cClk
I3C

enumerator kCLOCK_UsbClk
USB

enumerator kCLOCK_DmicClk
Digital Mic clock

enumerator kCLOCK_DspCpuClk
DSP clock

enumerator kCLOCK_AcmpClk
Acmp clock

enumerator kCLOCK_Flexcomm0Clk
Flexcomm0Clock

enumerator kCLOCK_Flexcomm1Clk
Flexcomm1Clock

enumerator kCLOCK_Flexcomm2Clk
Flexcomm2Clock

enumerator kCLOCK_Flexcomm3Clk
Flexcomm3Clock

enumerator kCLOCK_Flexcomm4Clk
Flexcomm4Clock

enumerator kCLOCK_Flexcomm5Clk
Flexcomm5Clock

enumerator kCLOCK_Flexcomm6Clk
Flexcomm6Clock

enumerator kCLOCK_Flexcomm7Clk
Flexcomm7Clock

enumerator kCLOCK_Flexcomm14Clk
Flexcomm14Clock

enumerator kCLOCK_Flexcomm15Clk
Flexcomm15Clock

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 _clock_attach_id
The enumerator of clock attach Id.
Values:

enumerator kSFRO_to_SYS_PLL
Attach SFRO to SYS_PLL.

enumerator kXTALIN_CLK_to_SYS_PLL
Attach XTALIN_CLK to SYS_PLL.

enumerator kFFRO_DIV2_to_SYS_PLL
Attach FFRO_DIV2 to SYS_PLL.

enumerator kNONE_to_SYS_PLL
Attach NONE to SYS_PLL.

enumerator kSFRO_to_AUDIO_PLL
Attach SFRO to AUDIO_PLL.

enumerator kXTALIN_CLK_to_AUDIO_PLL
Attach XTALIN_CLK to AUDIO_PLL.

enumerator kFFRO_DIV2_to_AUDIO_PLL
Attach FFRO_DIV2 to AUDIO_PLL.

enumerator kNONE_to_AUDIO_PLL
Attach NONE to AUDIO_PLL.

enumerator kFFRO_DIV4_to_MAIN_CLK
Attach FFRO_DIV4 to MAIN_CLK.

enumerator kXTALIN_CLK_to_MAIN_CLK
Attach XTALIN_CLK to MAIN_CLK.

enumerator kLPOSC_to_MAIN_CLK
Attach LPOSC to MAIN_CLK.

enumerator kFFRO_to_MAIN_CLK
Attach FFRO to MAIN_CLK.

enumerator kSFRO_to_MAIN_CLK
Attach SFRO to MAIN_CLK.

enumerator kMAIN_PLL_to_MAIN_CLK
Attach MAIN_PLL to MAIN_CLK.

enumerator kOSC32K_to_MAIN_CLK
Attach OSC32K to MAIN_CLK.

enumerator kFFRO_to_DSP_MAIN_CLK
Attach FFRO to DSP_MAIN_CLK.

enumerator kXTALIN_CLK_to_DSP_MAIN_CLK
Attach XTALIN_CLK to DSP_MAIN_CLK.

enumerator kLPOSC_to_DSP_MAIN_CLK
Attach LPOSC to DSP_MAIN_CLK.

enumerator kSFRO_to_DSP_MAIN_CLK
Attach SFRO to DSP_MAIN_CLK.

enumerator kMAIN_PLL_to_DSP_MAIN_CLK
Attach MAIN_PLL to DSP_MAIN_CLK.

enumerator kDSP_PLL_to_DSP_MAIN_CLK
Attach DSP_PLL to DSP_MAIN_CLK.

enumerator kOSC32K_to_DSP_MAIN_CLK
Attach OSC32K to DSP_MAIN_CLK.

enumerator kSFRO_to_ADC_CLK
Attach SFRO to ADC_CLK.

enumerator kXTALIN_CLK_to_ADC_CLK
Attach XTALIN_CLK to ADC_CLK.

enumerator kLPOSC_to_ADC_CLK
Attach LPOSC to ADC_CLK.

enumerator kFFRO_to_ADC_CLK
Attach FFRO to ADC_CLK.

enumerator kMAIN_PLL_to_ADC_CLK
Attach MAIN_PLL to ADC_CLK.

enumerator kAUX0_PLL_to_ADC_CLK
Attach AUX0_PLL to ADC_CLK.

enumerator kAUX1_PLL_to_ADC_CLK
Attach AUX1_PLL to ADC_CLK.

enumerator kSFRO_to_CLKOUT
Attach SFRO to CLKOUT.

enumerator kXTALIN_CLK_to_CLKOUT
Attach XTALIN_CLK to CLKOUT.

enumerator kLPOSC_to_CLKOUT
Attach LPOSC to CLKOUT.

enumerator kFFRO_to_CLKOUT
Attach FFRO to CLKOUT.

enumerator kMAIN_CLK_to_CLKOUT
Attach MAIN_CLK to CLKOUT.

enumerator kDSP_MAIN_to_CLKOUT
Attach DSP_MAIN to CLKOUT.

enumerator kMAIN_PLL_to_CLKOUT
Attach MAIN_PLL to CLKOUT.

enumerator kAUX0_PLL_to_CLKOUT
Attach AUX0_PLL to CLKOUT.

enumerator kDSP_PLL_to_CLKOUT
Attach DSP_PLL to CLKOUT.

enumerator kAUX1_PLL_to_CLKOUT
Attach AUX1_PLL to CLKOUT.

enumerator kAUDIO_PLL_to_CLKOUT
Attach AUDIO_PLL to CLKOUT.

enumerator kOSC32K_to_CLKOUT
Attach OSC32K to CLKOUT.

enumerator kNONE_to_CLKOUT
Attach NONE to CLKOUT.

enumerator kMAIN_CLK_to_I3C_CLK
Attach MAIN_CLK to I3C_CLK.

enumerator kFFRO_to_I3C_CLK
Attach FFRO to I3C_CLK.

enumerator kNONE_to_I3C_CLK
Attach NONE to I3C_CLK.

enumerator kI3C_CLK_to_I3C_TC_CLK
Attach I3C_CLK to I3C_TC_CLK.

enumerator kLPOSC_to_I3C_TC_CLK
Attach LPOSC to I3C_TC_CLK.

enumerator kNONE_to_I3C_TC_CLK
Attach NONE to I3C_TC_CLK.

enumerator kLPOSC_to_OSTIMER_CLK
Attach LPOSC to OSTIMER_CLK.

enumerator kOSC32K_to_OSTIMER_CLK
Attach OSC32K to OSTIMER_CLK.

enumerator kHCLK_to_OSTIMER_CLK
Attach HCLK to OSTIMER_CLK.

enumerator kNONE_to_OSTIMER_CLK
Attach NONE to OSTIMER_CLK.

enumerator kMAIN_CLK_to_FLEXSPI_CLK
Attach MAIN_CLK to FLEXSPI_CLK.

enumerator kMAIN_PLL_to_FLEXSPI_CLK
Attach MAIN_PLL to FLEXSPI_CLK.

enumerator kAUX0_PLL_to_FLEXSPI_CLK
Attach AUX0_PLL to FLEXSPI_CLK.

enumerator kFFRO_to_FLEXSPI_CLK
Attach FFRO to FLEXSPI_CLK.

enumerator kAUX1_PLL_to_FLEXSPI_CLK
Attach AUX1_PLL to FLEXSPI_CLK.

enumerator kNONE_to_FLEXSPI_CLK
Attach NONE to FLEXSPI_CLK.

enumerator kMAIN_CLK_to_SCT_CLK
Attach MAIN_CLK to SCT_CLK.

enumerator kMAIN_PLL_to_SCT_CLK
Attach MAIN_PLL to SCT_CLK.

enumerator kAUX0_PLL_to_SCT_CLK
Attach AUX0_PLL to SCT_CLK.

enumerator kFFRO_to_SCT_CLK
Attach FFRO to SCT_CLK.

enumerator kAUX1_PLL_to_SCT_CLK
Attach AUX1_PLL to SCT_CLK.

enumerator kAUDIO_PLL_to_SCT_CLK
Attach AUDIO_PLL to SCT_CLK.

enumerator kNONE_to_SCT_CLK
Attach NONE to SCT_CLK.

enumerator kLPOSC_to_UTICK_CLK
Attach LPOSC to UTICK_CLK.

enumerator kNONE_to_UTICK_CLK
Attach NONE to UTICK_CLK.

enumerator kLPOSC_to_WDT0_CLK
Attach LPOSC to WDT0_CLK.

enumerator kNONE_to_WDT0_CLK
Attach NONE to WDT0_CLK.

enumerator kLPOSC_to_WDT1_CLK
Attach LPOSC to WDT1_CLK.

enumerator kNONE_to_WDT1_CLK
Attach NONE to WDT1_CLK.

enumerator kOSC32K_to_32KHZWAKE_CLK
Attach OSC32K to 32KHZWAKE_CLK.

enumerator kLPOSC_DIV32_to_32KHZWAKE_CLK
Attach LPOSC_DIV32 to 32KHZWAKE_CLK.

enumerator kNONE_to_32KHZWAKE_CLK
Attach NONE to 32KHZWAKE_CLK.

enumerator kMAIN_CLK_DIV_to_SYSTICK_CLK
Attach MAIN_CLK_DIV to SYSTICK_CLK.

enumerator kLPOSC_to_SYSTICK_CLK
Attach LPOSC to SYSTICK_CLK.

enumerator kOSC32K_to_SYSTICK_CLK
Attach OSC32K to SYSTICK_CLK.

enumerator kSFRO_to_SYSTICK_CLK
Attach SFRO to SYSTICK_CLK.

enumerator kNONE_to_SYSTICK_CLK
Attach NONE to SYSTICK_CLK.

enumerator kMAIN_CLK_to_SDIO0_CLK
Attach MAIN_CLK to SDIO0_CLK.

enumerator kMAIN_PLL_to_SDIO0_CLK
Attach MAIN_PLL to SDIO0_CLK.

enumerator kAUX0_PLL_to_SDIO0_CLK
Attach AUX0_PLL to SDIO0_CLK.

enumerator kFFRO_to_SDIO0_CLK
Attach FFRO to SDIO0_CLK.

enumerator kAUX1_PLL_to_SDIO0_CLK
Attach AUX1_PLL to SDIO0_CLK.

enumerator kNONE_to_SDIO0_CLK
Attach NONE to SDIO0_CLK.

enumerator kMAIN_CLK_to_SDIO1_CLK
Attach MAIN_CLK to SDIO1_CLK.

enumerator kMAIN_PLL_to_SDIO1_CLK
Attach MAIN_PLL to SDIO1_CLK.

enumerator kAUX0_PLL_to_SDIO1_CLK
Attach AUX0_PLL to SDIO1_CLK.

enumerator kFFRO_to_SDIO1_CLK
Attach FFRO to SDIO1_CLK.

enumerator kAUX1_PLL_to_SDIO1_CLK
Attach AUX1_PLL to SDIO1_CLK.

enumerator kNONE_to_SDIO1_CLK
Attach NONE to SDIO1_CLK.

enumerator kFFRO_to_ESPI_CLK
Attach FFRO to ESPI_CLK.

enumerator kNONE_to_ESPI_CLK
Attach NONE to ESPI_CLK.

enumerator kXTALIN_CLK_to_USB_CLK
Attach XTALIN_CLK to USB_CLK.

enumerator kMAIN_CLK_to_USB_CLK
Attach MAIN_CLK to USB_CLK.

enumerator kNONE_to_USB_CLK
Attach NONE to USB_CLK.

enumerator kFFRO_to_MCLK_CLK
Attach FFRO to MCLK_CLK.

enumerator kAUDIO_PLL_to_MCLK_CLK
Attach AUDIO_PLL to MCLK_CLK.

enumerator kNONE_to_MCLK_CLK
Attach NONE to MCLK_CLK.

enumerator kSFRO_to_DMIC_CLK
Attach SFRO to DMIC_CLK.

enumerator kFFRO_to_DMIC_CLK
Attach FFRO to DMIC_CLK.

enumerator kAUDIO_PLL_to_DMIC_CLK
Attach AUDIO_PLL to DMIC_CLK.

enumerator kMASTER_CLK_to_DMIC_CLK
Attach MASTER_CLK to DMIC_CLK.

enumerator kLPOSC_to_DMIC_CLK
Attach LPOSC to DMIC_CLK.

enumerator k32K_WAKE_CLK_to_DMIC_CLK
Attach 32K_WAKE_CLK to DMIC_CLK.

enumerator kNONE_to_DMIC_CLK
Attach NONE to DMIC_CLK.

enumerator kMAIN_CLK_to_ACMP_CLK
Attach MAIN_CLK to ACMP_CLK.

enumerator kSFRO_to_ACMP_CLK
Attach SFRO to ACMP_CLK.

enumerator kFFRO_to_ACMP_CLK
Attach FFRO to ACMP_CLK.

enumerator kAUX0_PLL_to_ACMP_CLK
Attach AUX0_PLL to ACMP_CLK.

enumerator kAUX1_PLL_to_ACMP_CLK
Attach AUX1_PLL to ACMP_CLK.

enumerator kNONE_to_ACMP_CLK
Attach NONE to ACMP_CLK.

enumerator kSFRO_to_FLEXCOMM0
Attach SFRO to FLEXCOMM0.

enumerator kFFRO_to_FLEXCOMM0
Attach FFRO to FLEXCOMM0.

enumerator kAUDIO_PLL_to_FLEXCOMM0
Attach AUDIO_PLL to FLEXCOMM0.

enumerator kMASTER_CLK_to_FLEXCOMM0
Attach MASTER_CLK to FLEXCOMM0.

enumerator kFRG_to_FLEXCOMM0
Attach FRG to FLEXCOMM0.

enumerator kNONE_to_FLEXCOMM0
Attach NONE to FLEXCOMM0.

enumerator kSFRO_to_FLEXCOMM1
Attach SFRO to FLEXCOMM1.

enumerator kFFRO_to_FLEXCOMM1
Attach FFRO to FLEXCOMM1.

enumerator kAUDIO_PLL_to_FLEXCOMM1
Attach AUDIO_PLL to FLEXCOMM1.

enumerator kMASTER_CLK_to_FLEXCOMM1
Attach MASTER_CLK to FLEXCOMM1.

enumerator kFRG_to_FLEXCOMM1
Attach FRG to FLEXCOMM1.

enumerator kNONE_to_FLEXCOMM1
Attach NONE to FLEXCOMM1.

enumerator kSFRO_to_FLEXCOMM2
Attach SFRO to FLEXCOMM2.

enumerator kFFRO_to_FLEXCOMM2
Attach FFRO to FLEXCOMM2.

enumerator kAUDIO_PLL_to_FLEXCOMM2
Attach AUDIO_PLL to FLEXCOMM2.

enumerator kMASTER_CLK_to_FLEXCOMM2
Attach MASTER_CLK to FLEXCOMM2.

enumerator kFRG_to_FLEXCOMM2
Attach FRG to FLEXCOMM2.

enumerator kNONE_to_FLEXCOMM2
Attach NONE to FLEXCOMM2.

enumerator kSFRO_to_FLEXCOMM3
Attach SFRO to FLEXCOMM3.

enumerator kFFRO_to_FLEXCOMM3
Attach FFRO to FLEXCOMM3.

enumerator kAUDIO_PLL_to_FLEXCOMM3
Attach AUDIO_PLL to FLEXCOMM3.

enumerator kMASTER_CLK_to_FLEXCOMM3
Attach MASTER_CLK to FLEXCOMM3.

enumerator kFRG_to_FLEXCOMM3
Attach FRG to FLEXCOMM3.

enumerator kNONE_to_FLEXCOMM3
Attach NONE to FLEXCOMM3.

enumerator kSFRO_to_FLEXCOMM4
Attach SFRO to FLEXCOMM4.

enumerator kFFRO_to_FLEXCOMM4
Attach FFRO to FLEXCOMM4.

enumerator kAUDIO_PLL_to_FLEXCOMM4
Attach AUDIO_PLL to FLEXCOMM4.

enumerator kMASTER_CLK_to_FLEXCOMM4
Attach MASTER_CLK to FLEXCOMM4.

enumerator kFRG_to_FLEXCOMM4
Attach FRG to FLEXCOMM4.

enumerator kNONE_to_FLEXCOMM4
Attach NONE to FLEXCOMM4.

enumerator kSFRO_to_FLEXCOMM5
Attach SFRO to FLEXCOMM5.

enumerator kFFRO_to_FLEXCOMM5
Attach FFRO to FLEXCOMM5.

enumerator kAUDIO_PLL_to_FLEXCOMM5
Attach AUDIO_PLL to FLEXCOMM5.

enumerator kMASTER_CLK_to_FLEXCOMM5
Attach MASTER_CLK to FLEXCOMM5.

enumerator kFRG_to_FLEXCOMM5
Attach FRG to FLEXCOMM5.

enumerator kNONE_to_FLEXCOMM5
Attach NONE to FLEXCOMM5.

enumerator kSFRO_to_FLEXCOMM6
Attach SFRO to FLEXCOMM6.

enumerator kFFRO_to_FLEXCOMM6
Attach FFRO to FLEXCOMM6.

enumerator kAUDIO_PLL_to_FLEXCOMM6
Attach AUDIO_PLL to FLEXCOMM6.

enumerator kMASTER_CLK_to_FLEXCOMM6
Attach MASTER_CLK to FLEXCOMM6.

enumerator kFRG_to_FLEXCOMM6
Attach FRG to FLEXCOMM6.

enumerator kNONE_to_FLEXCOMM6
Attach NONE to FLEXCOMM6.

enumerator kSFRO_to_FLEXCOMM7
Attach SFRO to FLEXCOMM7.

enumerator kFFRO_to_FLEXCOMM7
Attach FFRO to FLEXCOMM7.

enumerator kAUDIO_PLL_to_FLEXCOMM7
Attach AUDIO_PLL to FLEXCOMM7.

enumerator kMASTER_CLK_to_FLEXCOMM7
Attach MASTER_CLK to FLEXCOMM7.

enumerator kFRG_to_FLEXCOMM7
Attach FRG to FLEXCOMM7.

enumerator kNONE_to_FLEXCOMM7
Attach NONE to FLEXCOMM7.

enumerator kSFRO_to_FLEXCOMM14
Attach SFRO to FLEXCOMM14.

enumerator kFFRO_to_FLEXCOMM14
Attach FFRO to FLEXCOMM14.

enumerator kAUDIO_PLL_to_FLEXCOMM14
Attach AUDIO_PLL to FLEXCOMM14.

enumerator kMASTER_CLK_to_FLEXCOMM14
Attach MASTER_CLK to FLEXCOMM14.

enumerator kFRG_to_FLEXCOMM14
Attach FRG to FLEXCOMM14.

enumerator kNONE_to_FLEXCOMM14
Attach NONE to FLEXCOMM14.

enumerator kSFRO_to_FLEXCOMM15
Attach SFRO to FLEXCOMM15.

enumerator kFFRO_to_FLEXCOMM15
Attach FFRO to FLEXCOMM15.

enumerator kAUDIO_PLL_to_FLEXCOMM15
Attach AUDIO_PLL to FLEXCOMM15.

enumerator kMASTER_CLK_to_FLEXCOMM15
Attach MASTER_CLK to FLEXCOMM15.

enumerator kFRG_to_FLEXCOMM15
Attach FRG to FLEXCOMM15.

enumerator kNONE_to_FLEXCOMM15
Attach NONE to FLEXCOMM15.

enumerator kMAIN_CLK_to_CTIMER0
Attach MAIN_CLK to CTIMER0.

enumerator kSFRO_to_CTIMER0
Attach SFRO to CTIMER0.

enumerator kFFRO_to_CTIMER0
Attach FFRO to CTIMER0.

enumerator kAUDIO_PLL_to_CTIMER0
Attach AUDIO_PLL to CTIMER0.

enumerator kMASTER_CLK_to_CTIMER0
Attach MASTER_CLK to CTIMER0.

enumerator kLPOSC_to_CTIMER0
Attach LPOSC to CTIMER0.

enumerator kNONE_to_CTIMER0
Attach NONE to CTIMER0.

enumerator kMAIN_CLK_to_CTIMER1
Attach MAIN_CLK to CTIMER1.

enumerator kSFRO_to_CTIMER1
Attach SFRO to CTIMER1.

enumerator kFFRO_to_CTIMER1
Attach FFRO to CTIMER1.

enumerator kAUDIO_PLL_to_CTIMER1
Attach AUDIO_PLL to CTIMER1.

enumerator kMASTER_CLK_to_CTIMER1
Attach MASTER_CLK to CTIMER1.

enumerator kLPOSC_to_CTIMER1
Attach LPOSC to CTIMER1.

enumerator kNONE_to_CTIMER1
Attach NONE to CTIMER1.

enumerator kMAIN_CLK_to_CTIMER2
Attach MAIN_CLK to CTIMER2.

enumerator kSFRO_to_CTIMER2
Attach SFRO to CTIMER2.

enumerator kFFRO_to_CTIMER2
Attach FFRO to CTIMER2.

enumerator kAUDIO_PLL_to_CTIMER2
Attach AUDIO_PLL to CTIMER2.

enumerator kMASTER_CLK_to_CTIMER2
Attach MASTER_CLK to CTIMER2.

enumerator kLPOSC_to_CTIMER2
Attach LPOSC to CTIMER2.

enumerator kNONE_to_CTIMER2
Attach NONE to CTIMER2.

enumerator kMAIN_CLK_to_CTIMER3
Attach MAIN_CLK to CTIMER3.

enumerator kSFRO_to_CTIMER3
Attach SFRO to CTIMER3.

enumerator kFFRO_to_CTIMER3
Attach FFRO to CTIMER3.

enumerator kAUDIO_PLL_to_CTIMER3
Attach AUDIO_PLL to CTIMER3.

enumerator kMASTER_CLK_to_CTIMER3
Attach MASTER_CLK to CTIMER3.

enumerator kLPOSC_to_CTIMER3
Attach LPOSC to CTIMER3.

enumerator kNONE_to_CTIMER3
Attach NONE to CTIMER3.

enumerator kMAIN_CLK_to_CTIMER4
Attach MAIN_CLK to CTIMER4.

enumerator kSFRO_to_CTIMER4
Attach SFRO to CTIMER4.

enumerator kFFRO_to_CTIMER4
Attach FFRO to CTIMER4.

enumerator kAUDIO_PLL_to_CTIMER4
Attach AUDIO_PLL to CTIMER4.

enumerator kMASTER_CLK_to_CTIMER4
Attach MASTER_CLK to CTIMER4.

enumerator kLPOSC_to_CTIMER4
Attach LPOSC to CTIMER4.

enumerator kNONE_to_CTIMER4
Attach NONE to CTIMER4.

enum _clock_div_name
Clock dividers.
Values:

enumerator kCLOCK_DivSysCpuAhbClk
Sys Cpu Ahb Clk Divider.

enumerator kCLOCK_DivMainPllClk
Main Pll Clk Divider.

enumerator kCLOCK_DivDspPllClk
Dsp Pll Clk Divider.

enumerator kCLOCK_DivAux0PllClk
Aux0 Pll Clk Divider.

enumerator kCLOCK_DivAux1PllClk
Aux1 Pll Clk Divider.

enumerator kCLOCK_DivPfc0Clk
Pfc0 Clk Divider.

enumerator kCLOCK_DivPfc1Clk
Pfc1 Clk Divider.

enumerator kCLOCK_DivAdcClk
Adc Clk Divider.

enumerator kCLOCK_DivFlexspiClk
Flexspi Clk Divider.

enumerator kCLOCK_DivSctClk
Sct Clk Divider.

enumerator kCLOCK_DivSdio0Clk
Sdio0 Clk Divider.

enumerator kCLOCK_DivSdio1Clk
Sdio1 Clk Divider.

enumerator kCLOCK_DivSystickClk
Systick Clk Divider.

enumerator kCLOCK_DivUsbHsFclk
Usb Hs Fclk Divider.

enumerator kCLOCK_DivAudioPllClk
Audio Pll Clk Divider.

enumerator kCLOCK_DivAcmpClk
Acmp Clk Divider.

enumerator kCLOCK_DivClockOut
Clock Out Divider.

enumerator kCLOCK_DivDmicClk
Dmic Clk Divider.

enumerator kCLOCK_DivDspCpuClk
Dsp Cpu Clk Divider.

enumerator kCLOCK_DivDspRamClk
Dsp Ram Clk Divider.

enumerator kCLOCK_DivMclkClk
Mclk Clk Divider.

enumerator kCLOCK_DivPllFrgClk
Pll Frg Clk Divider.

enumerator kCLOCK_DivI3cClk
I3c Clk Divider.

enumerator kCLOCK_DivI3cTcClk
I3c Tc Clk Divider.

enumerator kCLOCK_DivI3cSlowClk
I3c Slow Clk Divider.

enum _clock_ffro_freq
FFRO frequence configuration.
Values:

enumerator kCLOCK_Ffro48M
48MHz FFRO clock.

enumerator kCLOCK_Ffro60M
60MHz FFRO clock.

enum _sys_pll_src
SysPLL Reference Input Clock Source.
Values:

enumerator kCLOCK_SysPllSFroClk
16MHz FRO clock

enumerator kCLOCK_SysPllXtalIn
OSC clock

enumerator kCLOCK_SysPllFFroDiv2
FRO48/60 div2 clock

enumerator kCLOCK_SysPllNone
Gated to reduce power

enum _sys_pll_mult
SysPLL Multiplication Factor.
Values:

enumerator kCLOCK_SysPllMult16
Divide by 16

enumerator kCLOCK_SysPllMult17
Divide by 17

enumerator kCLOCK_SysPllMult18
Divide by 18

enumerator kCLOCK_SysPllMult19
Divide by 19

enumerator kCLOCK_SysPllMult20
Divide by 20

enumerator kCLOCK_SysPllMult21
Divide by 21

enumerator kCLOCK_SysPllMult22
Divide by 22

enum _audio_pll_src
AudioPll Reference Input Clock Source.
Values:

enumerator kCLOCK_AudioPllSFroClk
16MHz FRO clock

enumerator kCLOCK_AudioPllXtalIn
OSC clock

enumerator kCLOCK_AudioPllFFroDiv2
FRO48/60 div2 clock

enumerator kCLOCK_AudioPllNone
Gated to reduce power

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

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_pfd* clock_pfd_t
PLL PFD clock name.

typedef enum *_clock_attach_id* clock_attach_id_t
The enumerator of clock attach Id.

typedef enum *_clock_div_name* clock_div_name_t
Clock dividers.

typedef enum *_clock_ffro_freq* clock_ffro_freq_t
FFRO frequence configuration.

typedef enum *_sys_pll_src* sys_pll_src_t
SysPLL Reference Input Clock Source.

typedef enum *_sys_pll_mult* sys_pll_mult_t
SysPLL Multiplication Factor.

typedef struct *_clock_sys_pll_config* clock_sys_pll_config_t
PLL configuration for SYSPLL.

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 SYSPLL.

typedef struct *_clock_frg_clk_config* clock_frg_clk_config_t

PLL configuration for FRG.

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) clock frequency.

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_mclkFreq

External XTAL (SYSOSC) clock frequency.

The MCLK in (mclk_in) PIN clock frequency in Hz, when the clock is setup, use the function CLOCK_SetMclkInFreq to set the value in to clock driver. For example, if mclk_In is 16MHz,

```
CLOCK_SetMclkInFreq(160000000);
```

static inline void CLOCK_EnableClock(*clock_ip_name_t* clk)

static inline void CLOCK_DisableClock(*clock_ip_name_t* clk)

void CLOCK_AttachClk(*clock_attach_id_t* connection)

Configure the clock selection muxes.

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_GetFRGClock(uint32_t id)

Return Input frequency for the Fractional baud rate generator.

Returns

Input Frequency for FRG

void CLOCK_SetFRGClock(const *clock_frg_clk_config_t* *config)

Set output of the Fractional baud rate generator.

Parameters

- config – : Configuration to set to FRGn clock.

static inline uint32_t CLOCK_GetSFroFreq(void)

Return Frequency of FRO 16MHz.

Returns

Frequency of FRO 16MHz

uint32_t CLOCK_GetSysPllFreq(void)

Return Frequency of SYSPLL.

Returns

Frequency of SYSPLL

uint32_t CLOCK_GetSysPfdFreq(*clock_pfd_t* pfd)

Get current output frequency of specific System PLL PFD.

Parameters

- pfd – : pfd name to get frequency.

Returns

Frequency of SYSPLL 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.

uint32_t CLOCK_GetFFroFreq(void)

Return Frequency of High-Freq output of FRO.

Returns

Frequency of High-Freq output of FRO

uint32_t CLOCK_GetMainClkFreq(void)

Return Frequency of main clk.

Returns

Frequency of main clk

uint32_t CLOCK_GetDspMainClkFreq(void)

Return Frequency of DSP main clk.

Returns

Frequency of DSP main clk

uint32_t CLOCK_GetAcmpClkFreq(void)

Return Frequency of ACMP clk.

Returns
Frequency of ACMP clk

uint32_t CLOCK_GetDmicClkFreq(void)

Return Frequency of DMIC clk.

Returns
Frequency of DMIC clk

uint32_t CLOCK_GetUsbClkFreq(void)

Return Frequency of USB clk.

Returns
Frequency of USB clk

uint32_t CLOCK_GetSdioClkFreq(uint32_t id)

Return Frequency of SDIO clk.

Parameters

- id – : SDIO index to get frequency.

Returns
Frequency of SDIO clk

uint32_t CLOCK_GetI3cClkFreq(void)

Return Frequency of I3C clk.

Returns
Frequency of I3C clk

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_GetMclkClkFreq(void)

Return Frequency of mclk.

Returns
Frequency of mclk clk

uint32_t CLOCK_GetSctClkFreq(void)

Return Frequency of sct.

Returns
Frequency of sct 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.

static inline uint32_t CLOCK_GetXtalInClkFreq(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.

static inline uint32_t CLOCK_GetLpOscFreq(void)

Return Frequency of Lower power osc.

Returns

Frequency of LPOSC

static inline uint32_t CLOCK_GetOsc32KFreq(void)

Return Frequency of 32kHz osc.

Returns

Frequency of 32kHz osc

static inline void CLOCK_EnableOsc32K(bool enable)

Enables and disables 32kHz osc.

Parameters

- enable – : true to enable 32k osc clock, false to disable clock

static inline 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.

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_SetMclkFreq(uint32_t freq)

Set the MCLK in (mclk_in) clock frequency based on board setting.

Parameters

- freq – : The MCLK input clock frequency in Hz.

uint32_t CLOCK_GetFlexCommClkFreq(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_GetCtimerClkFreq(uint32_t id)
Return Frequency of Ctimer Clock.

Parameters
- id – : ctimer index to get frequency.

Returns
Frequency of Ctimer Clock

uint32_t CLOCK_GetClockOutClkFreq(void)
Return Frequency of ClockOut.

Returns
Frequency of ClockOut

uint32_t CLOCK_GetAdcClkFreq(void)
Return Frequency of Adc Clock.

Returns
Frequency of Adc Clock.

uint32_t CLOCK_GetFlexspiClkFreq(void)
Return Frequency of Flexspi Clock.

Returns
Frequency of Flexspi.

void CLOCK_EnableFfroClk(*clock_ffro_freq_t* ffroFreq)
brief Enable FFRO 48M/60M clock. param ffroFreq : target fro frequency. return Nothing

void CLOCK_EnableSfroClk(void)
brief Enable SFRO clock. param Nothing return Nothing

void CLOCK_InitSysPll(const *clock_sys_pll_config_t* *config)
Initialize the System PLL.

Parameters
- config – : Configuration to set to PLL.

static inline void CLOCK_DeinitSysPll(void)
brief Deinit the System PLL. param none.

status_t CLOCK_InitSysPfd(*clock_pfd_t* pfd, uint8_t divider)
Initialize the System 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_DeinitSysPfd(*clock_pfd_t* pfd)
brief Disable the audio PLL PFD. param 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)

brief Deinit the Audio PLL. param 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(uint32_t pfd)

brief Disable the audio PLL PFD. param pfd : Which PFD clock to disable.

void CLOCK_EnableUsbhsDeviceClock(void)

Enable USB HS device PLL clock.

This function enables USB HS device PLL clock.

void CLOCK_EnableUsbhsHostClock(void)

Enable USB HS host PLL clock.

This function enables USB HS host PLL clock.

bool CLOCK_EnableUsbHs0PhyPllClock(*clock_attach_id_t* src, uint32_t freq)

brief Enable USB hs0PhyPll clock.

param src USB HS clock source. param freq The frequency specified by src. retval true The clock is set successfully. retval false The clock source is invalid to get proper USB HS clock.

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.7.4.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

CLOCK_GetCTimerClkFreq

CMP_CLOCKS

Clock ip name array for ACMP.

FLEXCOMM_CLOCKS

Clock ip name array for FLEXCOMM.

USART_CLOCKS

Clock ip name array for LPUART.

I2C_CLOCKS
: Clock ip name array for I2C.

I3C_CLOCKS
: Clock ip name array for I3C.

SPI_CLOCKS
: Clock ip name array for SPI.

FLEXI2S_CLOCKS
: Clock ip name array for FLEXI2S.

UTICK_CLOCKS
: Clock ip name array for UTICK.

DMIC_CLOCKS
: Clock ip name array for DMIC.

CTIMER_CLOCKS
: Clock ip name array for CT32B.

GPIO_CLOCKS
: Clock ip name array for GPIO.

LPADC_CLOCKS
: Clock ip name array for ADC.

MRT_CLOCKS
: Clock ip name array for MRT.

SCT_CLOCKS
: Clock ip name array for SCT.

RTC_CLOCKS
: Clock ip name array for RTC.

WWDT_CLOCKS
: Clock ip name array for WWDT.

CRC_CLOCKS
: Clock ip name array for CRC.

USBD_CLOCKS
: Clock ip name array for USBD.

DMA_CLOCKS
: Clock ip name array for DMA.

PINT_CLOCKS
: Clock ip name array for PINT.

FLEXSPI_CLOCKS
: Clock ip name array for FLEXSPI.

CACHE64_CLOCKS
: Clock ip name array for Cache64.

MU_CLOCKS
: Clock ip name array for MUA.

SEMA42_CLOCKS
: Clock ip name array for SEMA.

TRNG_CLOCKS
: Clock ip name array for RNG.

PUF_CLOCKS
: Clock ip name array for PUF.

HASHCRYPT_CLOCKS
: Clock ip name array for HashCrypt.

CASPER_CLOCKS
: Clock ip name array for Casper.

USDHC_CLOCKS
: Clock ip name array for uSDHC.

OSTIMER_CLOCKS
: Clock ip name array for OSTimer.

POWERQUAD_CLOCKS
: Clock ip name array for Powerquad.

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_CTL1_PSCCTL0

CLK_CTL1_PSCCTL1

CLK_CTL1_PSCCTL2

SYSPLL0CLKSEL_OFFSET

Clock Mux Switches The encoding is as follows each connection identified is 32bits wide starting from LSB upwards.

[12 bits for reg offset, 0 means end of descriptor, 4 bits for choice] [bit 31 define CLKCTL0 or CLKCTL1]*

MAINCLKSELA_OFFSET

MAINCLKSELB_OFFSET

FLEXSPIFCLKSEL_OFFSET

SCTFCLKSEL_OFFSET

USBHSFCLKSEL_OFFSET

SDIO0FCLKSEL_OFFSET

SDIO1FCLKSEL_OFFSET

ESPICLKSEL_OFFSET

ADC0FCLKSEL0_OFFSET

ADC0FCLKSEL1_OFFSET

UTICKFCLKSEL_OFFSET

WDT0FCLKSEL_OFFSET

WAKECLK32KHZSEL_OFFSET

SYSTICKFCLKSEL_OFFSET

AUDIOPLL0CLKSEL_OFFSET

DSPCPUCLKSELA_OFFSET

DSPCPUCLKSELB_OFFSET

OSEVENTFCLKSEL_OFFSET

FC0FCLKSEL_OFFSET

FC1FCLKSEL_OFFSET

FC2FCLKSEL_OFFSET

FC3FCLKSEL_OFFSET

FC4FCLKSEL_OFFSET

FC5FCLKSEL_OFFSET

FC6FCLKSEL_OFFSET

FC7FCLKSEL_OFFSET

FC14FCLKSEL_OFFSET

FC15FCLKSEL_OFFSET

DMIC0FCLKSEL_OFFSET

CT32BIT0FCLKSEL_OFFSET

CT32BIT1FCLKSEL_OFFSET

CT32BIT2FCLKSEL_OFFSET

CT32BIT3FCLKSEL_OFFSET

CT32BIT4FCLKSEL_OFFSET

AUDIOMCLKSEL_OFFSET

CLKOUTSEL0_OFFSET

CLKOUTSEL1_OFFSET

I3C0FCLKSEL_OFFSET

I3C0FCLKSTCSEL_OFFSET

WDT1FCLKSEL_OFFSET

ACMP0FCLKSEL_OFFSET

MAINPLLCLKDIV_OFFSET

DSPPLLCLKDIV_OFFSET

AUX0PLLCLKDIV_OFFSET

AUX1PLLCLKDIV_OFFSET

SYSCPUAHBCLKDIV_OFFSET

PFC0CLKDIV_OFFSET

PFC1CLKDIV_OFFSET

FLEXSPIFCLKDIV_OFFSET

SCTFCLKDIV_OFFSET

USBHSFCLKDIV_OFFSET

SDIO0FCLKDIV_OFFSET

SDIO1FCLKDIV_OFFSET

ADC0FCLKDIV_OFFSET

WAKECLK32KHZDIV_OFFSET

SYSTICKFCLKDIV_OFFSET

AUDIOPLLCLKDIV_OFFSET

DSPCPUCLKDIV_OFFSET

DSPMAINRAMCLKDIV_OFFSET

FRGPLLCLKDIV_OFFSET

DMIC0FCLKDIV_OFFSET

AUDIOMCLKDIV_OFFSET

CLKOUTDIV_OFFSET

I3C0FCLKSTCDIV_OFFSET

I3C0FCLKSDIV_OFFSET

I3C0FCLKDIV_OFFSET

ACMP0FCLKDIV_OFFSET

CLKCTL0_TUPLE_MUXA(**reg**, **choice**)

CLKCTL0_TUPLE_MUXB(**reg**, **choice**)

CLKCTL1_TUPLE_MUXA(**reg**, **choice**)

CLKCTL1_TUPLE_MUXB(**reg**, **choice**)

CLKCTL_TUPLE_REG(**base**, **tuple**)

CLKCTL_TUPLE_SEL(**tuple**)

Values:

enumerator kCLOCK_FrgMainClk
Main System clock

enumerator kCLOCK_FrgPllDiv
Main pll clock divider

enumerator kCLOCK_FrgSFro
16MHz FRO

enumerator kCLOCK_FrgFFro
FRO48/60

sys_pll_src_t sys_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.

sys_pll_mult_t sys_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

uint8_t num
: FRG clock

enum *_clock_frg_clk_config* sfg_clock_src

uint8_t divider
: Denominator of the fractional divider.

uint8_t mult
: Numerator of the fractional divider.

struct _clock_sys_pll_config
: *#include <fsl_clock.h>* PLL configuration for SYSPLL.

struct _clock_audio_pll_config
: *#include <fsl_clock.h>* PLL configuration for SYSPLL.

struct _clock_frg_clk_config
: *#include <fsl_clock.h>* PLL configuration for FRG.

2.7 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION
: CRC driver version. Version 2.1.1.

  Current version: 2.1.1

  Change log:

  - Version 2.0.0
    - initial version
  - Version 2.0.1
    - add explicit type cast when writing to WR_DATA

- Version 2.0.2
  - Fix MISRA issue
- Version 2.1.0
  - Add CRC_WriteSeed function
- Version 2.1.1
  - Fix MISRA issue

enum _crc_polynomial

CRC polynomials to use.

Values:

enumerator kCRC_Polynomial_CRC_CCITT

x^16+x^12+x^5+1

enumerator kCRC_Polynomial_CRC_16

x^16+x^15+x^2+1

enumerator kCRC_Polynomial_CRC_32

x^32+x^26+x^23+x^22+x^16+x^12+x^11+x^10+x^8+x^7+x^5+x^4+x^2+x+1

typedef enum *_crc_polynomial* crc_polynomial_t

CRC polynomials to use.

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 functions enables the CRC peripheral clock in the LPC SYSCON block. It also configures the CRC engine and starts checksum computation by writing the seed.

Parameters

- base – CRC peripheral address.
- config – CRC module configuration structure.

static inline void CRC_Deinit(CRC_Type *base)

Disables the CRC peripheral module.

This functions disables the CRC peripheral clock in the LPC SYSCON block.

Parameters

- base – CRC peripheral address.

void CRC_Reset(CRC_Type *base)

resets CRC peripheral module.

Parameters

- base – CRC peripheral address.

void CRC_WriteSeed(CRC_Type *base, uint32_t seed)

Write seed to CRC peripheral module.

Parameters

- base – CRC peripheral address.
- seed – CRC Seed value.

void CRC_GetDefaultConfig(*crc_config_t* *config)

Loads default values to CRC protocol configuration structure.

Loads default values to CRC protocol configuration structure. The default values are:

```
config->polynomial = kCRC_Polynomial_CRC_CCITT;
config->reverseIn = false;
config->complementIn = false;
config->reverseOut = false;
config->complementOut = false;
config->seed = 0xFFFFU;
```

Parameters

- config – CRC protocol configuration structure

void CRC_GetConfig(CRC_Type *base, *crc_config_t* *config)

Loads actual values configured in CRC peripheral to CRC protocol configuration structure.

The values, including seed, can be used to resume CRC calculation later.

Parameters

- base – CRC peripheral address.
- 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 CRC data register.

Parameters

- base – CRC peripheral address.
- data – Input data stream, MSByte in data[0].
- dataSize – Size of the input data buffer in bytes.

static inline uint32_t CRC_Get32bitResult(CRC_Type *base)

Reads 32-bit checksum from the CRC module.

Reads CRC data register.

Parameters

- base – CRC peripheral address.

Returns

final 32-bit checksum, after configured bit reverse and complement operations.

static inline uint16_t CRC_Get16bitResult(CRC_Type *base)

Reads 16-bit checksum from the CRC module.

Reads CRC data register.

Parameters

- base – CRC peripheral address.

Returns
final 16-bit checksum, after configured bit reverse and complement operations.

CRC_DRIVER_USE_CRC16_CCITT_FALSE_AS_DEFAULT
Default configuration structure filled by *CRC_GetDefaultConfig()*. Uses CRC-16/CCITT-FALSE as default.

struct _crc_config
#include <fsl_crc.h> CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

Public Members

crc_polynomial_t polynomial
CRC polynomial.

bool reverseIn
Reverse bits on input.

bool complementIn
Perform 1's complement on input.

bool reverseOut
Reverse bits on output.

bool complementOut
Perform 1's complement on output.

uint32_t seed
Starting checksum value.

2.8 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

2.9 DMA: Direct Memory Access Controller Driver

void DMA_Init(DMA_Type *base)

Initializes DMA peripheral.

This function enable the DMA clock, set descriptor table and enable DMA peripheral.

Parameters

- base – DMA peripheral base address.

void DMA_Deinit(DMA_Type *base)

Deinitializes DMA peripheral.

This function gates the DMA clock.

Parameters

- base – DMA peripheral base address.

void DMA_InstallDescriptorMemory(DMA_Type *base, void *addr)

Install DMA descriptor memory.

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, althrough current DMA driver has a default DMA descriptor buffer, but it support one DMA descriptor for one channel only.

Parameters

- base – DMA base address.
- addr – DMA descriptor address

static inline bool DMA_ChannelIsActive(DMA_Type *base, uint32_t channel)

Return whether DMA channel is processing transfer.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

Returns

True for active state, false otherwise.

static inline bool DMA_ChannelIsBusy(DMA_Type *base, uint32_t channel)

Return whether DMA channel is busy.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

Returns

True for busy state, false otherwise.

static inline void DMA_EnableChannelInterrupts(DMA_Type *base, uint32_t channel)

Enables the interrupt source for the DMA transfer.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_DisableChannelInterrupts(DMA_Type *base, uint32_t channel)

Disables the interrupt source for the DMA transfer.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_EnableChannel(DMA_Type *base, uint32_t channel)

Enable DMA channel.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_DisableChannel(DMA_Type *base, uint32_t channel)

Disable DMA channel.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_EnableChannelPeriphRq(DMA_Type *base, uint32_t channel)

Set PERIPHREQEN of channel configuration register.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_DisableChannelPeriphRq(DMA_Type *base, uint32_t channel)

Get PERIPHREQEN value of channel configuration register.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

Returns

True for enabled PeriphRq, false for disabled.

void DMA_ConfigureChannelTrigger(DMA_Type *base, uint32_t channel, *dma_channel_trigger_t* *trigger)

Set trigger settings of DMA channel.

Deprecated:

Do not use this function. It has been superceded by *DMA_SetChannelConfig*.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

- trigger – trigger configuration.

void DMA_SetChannelConfig(DMA_Type *base, uint32_t channel, *dma_channel_trigger_t* *trigger, bool isPeriph)

set channel config.

This function provide a interface to configure channel configuration reisters.

Parameters

- base – DMA base address.
- channel – DMA channel number.
- trigger – channel configurations structure.
- isPeriph – true is periph request, false is not.

static inline uint32_t DMA_SetChannelXferConfig(bool reload, bool clrTrig, bool intA, bool intB, uint8_t width, uint8_t srcInc, uint8_t dstInc, uint32_t bytes)

DMA channel xfer transfer configurations.

Parameters

- reload – true is reload link descriptor after current exhaust, false is not
- clrTrig – true is clear trigger status, wait software trigger, false is not
- intA – enable interruptA
- intB – enable interruptB
- width – transfer width
- srcInc – source address interleave size
- dstInc – destination address interleave size
- bytes – transfer bytes

Returns

The vaule of xfer config

uint32_t DMA_GetRemainingBytes(DMA_Type *base, uint32_t channel)

Gets the remaining bytes of the current DMA descriptor transfer.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

Returns

The number of bytes which have not been transferred yet.

static inline void DMA_SetChannelPriority(DMA_Type *base, uint32_t channel, *dma_priority_t* priority)

Set priority of channel configuration register.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.
- priority – Channel priority value.

static inline *dma_priority_t* DMA_GetChannelPriority(DMA_Type *base, uint32_t channel)

Get priority of channel configuration register.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

Returns

Channel priority value.

static inline void DMA_SetChannelConfigValid(DMA_Type *base, uint32_t channel)

Set channel configuration valid.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_DoChannelSoftwareTrigger(DMA_Type *base, uint32_t channel)

Do software trigger for the channel.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.

static inline void DMA_LoadChannelTransferConfig(DMA_Type *base, uint32_t channel, uint32_t xfer)

Load channel transfer configurations.

Parameters

- base – DMA peripheral base address.
- channel – DMA channel number.
- xfer – transfer configurations.

void DMA_CreateDescriptor(*dma_descriptor_t* *desc, *dma_xfercfg_t* *xfercfg, void *srcAddr, void *dstAddr, void *nextDesc)

Create application specific DMA descriptor to be used in a chain in transfer.

Deprecated:

Do not use this function. It has been superceded by *DMA_SetupDescriptor*.

Parameters

- desc – DMA descriptor address.
- xfercfg – Transfer configuration for DMA descriptor.
- srcAddr – Address of last item to transmit
- dstAddr – Address of last item to receive.
- nextDesc – Address of next descriptor in chain.

void DMA_SetupDescriptor(*dma_descriptor_t* *desc, uint32_t xfercfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc)

setup dma descriptor

Note: This function do not support configure wrap descriptor.

Parameters

- desc – DMA descriptor address.
- xfercfg – Transfer configuration for DMA descriptor.
- srcStartAddr – Start address of source address.
- dstStartAddr – Start address of destination address.
- nextDesc – Address of next descriptor in chain.

void DMA_SetupChannelDescriptor(*dma_descriptor_t* *desc, uint32_t xfercfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc, *dma_burst_wrap_t* wrapType, uint32_t burstSize)

setup dma channel descriptor

Note: This function support configure wrap descriptor.

Parameters

- desc – DMA descriptor address.
- xfercfg – Transfer configuration for DMA descriptor.
- srcStartAddr – Start address of source address.
- dstStartAddr – Start address of destination address.
- nextDesc – Address of next descriptor in chain.
- wrapType – burst wrap type.
- burstSize – burst size, reference _dma_burst_size.

void DMA_LoadChannelDescriptor(DMA_Type *base, uint32_t channel, *dma_descriptor_t* *descriptor)

load channel transfer decriptor.

This function can be used to load desscriptor to driver internal channel descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

a. for the polling transfer, application can allocate a local descriptor memory table to prepare a descriptor firstly and then call this api to load the configured descriptor to driver descriptor table.

```
DMA_Init(DMA0);
DMA_EnableChannel(DMA0, DEMO_DMA_CHANNEL);
DMA_SetupDescriptor(desc, xferCfg, s_srcBuffer, &s_destBuffer[0], NULL);
DMA_LoadChannelDescriptor(DMA0, DEMO_DMA_CHANNEL, (dma_descriptor_t *)desc);
DMA_DoChannelSoftwareTrigger(DMA0, DEMO_DMA_CHANNEL);
while(DMA_ChannelIsBusy(DMA0, DEMO_DMA_CHANNEL))
{}
```

Parameters

- base – DMA base address.
- channel – DMA channel.
- descriptor – configured DMA descriptor.

void DMA_AbortTransfer(*dma_handle_t* *handle)

Abort running transfer by handle.

This function aborts DMA transfer specified by handle.

Parameters

- handle – DMA handle pointer.

void DMA_CreateHandle(*dma_handle_t* *handle, DMA_Type *base, uint32_t channel)

Creates the DMA handle.

This function is called if using transaction API for DMA. This function initializes the internal state of DMA handle.

Parameters

- handle – DMA handle pointer. The DMA handle stores callback function and parameters.
- base – DMA peripheral base address.
- channel – DMA channel number.

void DMA_SetCallback(*dma_handle_t* *handle, *dma_callback* callback, void *userData)

Installs a callback function for the DMA transfer.

This callback is called in DMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

Parameters

- handle – DMA handle pointer.
- callback – DMA callback function pointer.
- userData – Parameter for callback function.

void DMA_PrepareTransfer(*dma_transfer_config_t* *config, void *srcAddr, void *dstAddr, uint32_t byteWidth, uint32_t transferBytes, *dma_transfer_type_t* type, void *nextDesc)

Prepares the DMA transfer structure.

Deprecated:

Do not use this function. It has been superceded by *DMA_PrepareChannelTransfer*. 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, so the source address must be 4 bytes aligned, or it shall result in source address error(SAE).

Parameters

- config – The user configuration structure of type dma_transfer_t.
- srcAddr – DMA transfer source address.
- dstAddr – DMA transfer destination address.
- byteWidth – DMA transfer destination address width(bytes).
- transferBytes – DMA transfer bytes to be transferred.
- type – DMA transfer type.
- nextDesc – Chain custom descriptor to transfer.

void DMA_PrepareChannelTransfer(*dma_channel_config_t* *config, void *srcStartAddr, void *dstStartAddr, uint32_t xferCfg, *dma_transfer_type_t* type, *dma_channel_trigger_t* *trigger, void *nextDesc)

Prepare channel transfer configurations.

This function used to prepare channel transfer configurations.

Parameters

- config – Pointer to DMA channel transfer configuration structure.
- srcStartAddr – source start address.
- dstStartAddr – destination start address.
- xferCfg – xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value.
- type – transfer type.
- trigger – DMA channel trigger configurations.
- nextDesc – address of next descriptor.

status_t DMA_SubmitTransfer(*dma_handle_t* *handle, *dma_transfer_config_t* *config)

Submits the DMA transfer request.

Deprecated:

Do not use this function. It has been superceded by *DMA_SubmitChannelTransfer*.

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

Parameters

- handle – DMA handle pointer.
- config – Pointer to DMA transfer configuration structure.

Return values

- kStatus_DMA_Success – It means submit transfer request succeed.
- kStatus_DMA_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus_DMA_Busy – It means the given channel is busy, need to submit request later.

void DMA_SubmitChannelTransferParameter(*dma_handle_t* *handle, uint32_t xferCfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc)

Submit channel transfer paramter directly.

This function used to configue channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

a. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelTransferParameter(handle, DMA_CHANNEL_XFER(reload, clrTrig,␣
↪intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, NULL);
DMA_StartTransfer(handle)
```

b. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
  define link descriptor table in application with macro
  DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[3]);

  DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
  DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
  DMA_SetupDescriptor(nextDesc2,  DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
  DMA_SetChannelConfig(base, channel, trigger, isPeriph);
  DMA_CreateHandle(handle, base, channel)
  DMA_SubmitChannelTransferParameter(handle, DMA_CHANNEL_XFER(reload, clrTrig,␣
↪intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, nextDesc0);
  DMA_StartTransfer(handle);
```

Parameters

- handle – Pointer to DMA handle.
- xferCfg – xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value.
- srcStartAddr – source start address.
- dstStartAddr – destination start address.
- nextDesc – address of next descriptor.

void DMA_SubmitChannelDescriptor(*dma_handle_t* *handle, *dma_descriptor_t* *descriptor)

Submit channel descriptor.

This function used to configue channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, this functiono is typical for the ping pong case:

a. for the ping pong case, application should responsible for the descriptor, for example, application should prepare two descriptor table with macro.

```
  define link descriptor table in application with macro
  DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[2]);

  DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
  DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
  DMA_SetChannelConfig(base, channel, trigger, isPeriph);
  DMA_CreateHandle(handle, base, channel)
  DMA_SubmitChannelDescriptor(handle,  nextDesc0);
  DMA_StartTransfer(handle);
```

Parameters

- handle – Pointer to DMA handle.
- descriptor – descriptor to submit.

status_t DMA_SubmitChannelTransfer(*dma_handle_t* *handle, *dma_channel_config_t* *config)

Submits the DMA channel transfer request.

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time. It is used for the case:

a. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
DMA_CreateHandle(handle, base, channel)
DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,trigger,NULL);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)
```

b. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
  define link descriptor table in application with macro
  DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);
  DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
  DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
  DMA_SetupDescriptor(nextDesc2,  DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
  DMA_CreateHandle(handle, base, channel)
  DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,trigger,
↪nextDesc0);
  DMA_SubmitChannelTransfer(handle, config)
  DMA_StartTransfer(handle)
```

c. for the ping pong case, application should responsible for link descriptor, for example, application should prepare two descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
  define link descriptor table in application with macro
  DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);

  DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
  DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
↪ srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
  DMA_CreateHandle(handle, base, channel)
  DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,trigger,
↪nextDesc0);
  DMA_SubmitChannelTransfer(handle, config)
  DMA_StartTransfer(handle)
```

Parameters

- handle – DMA handle pointer.
- config – Pointer to DMA transfer configuration structure.

Return values

- kStatus_DMA_Success – It means submit transfer request succeed.
- kStatus_DMA_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus_DMA_Busy – It means the given channel is busy, need to submit request later.

void DMA_StartTransfer(*dma_handle_t* *handle)

DMA start transfer.

This function enables the channel request. User can call this function after submitting the transfer request It will trigger transfer start with software trigger only when hardware trigger is not used.

Parameters

- handle – DMA handle pointer.

void DMA_IRQHandle(DMA_Type *base)

DMA IRQ handler for descriptor transfer complete.

This function clears the channel major interrupt flag and call the callback function if it is not NULL.

Parameters

- base – DMA base address.

void DMA_DriverIRQHandler(uint32_t instance)

DMA common IRQ handler entry for parameterized dispatch.

Parameters

- instance – DMA peripheral instance number.

FSL_DMA_DRIVER_VERSION

DMA driver version.

Version 2.5.5.

_dma_transfer_status DMA transfer status

Values:

enumerator kStatus_DMA_Busy

Channel is busy and can't handle the transfer request.

_dma_addr_interleave_size dma address interleave size

Values:

enumerator kDMA_AddressInterleave0xWidth

dma source/destination address no interleave

enumerator kDMA_AddressInterleave1xWidth

dma source/destination address interleave 1xwidth

enumerator kDMA_AddressInterleave2xWidth
dma source/destination address interleave 2xwidth

enumerator kDMA_AddressInterleave4xWidth
dma source/destination address interleave 3xwidth

_dma_transfer_width dma transfer width

Values:

enumerator kDMA_Transfer8BitWidth
dma channel transfer bit width is 8 bit

enumerator kDMA_Transfer16BitWidth
dma channel transfer bit width is 16 bit

enumerator kDMA_Transfer32BitWidth
dma channel transfer bit width is 32 bit

enum _dma_priority
DMA channel priority.

Values:

enumerator kDMA_ChannelPriority0
Highest channel priority - priority 0

enumerator kDMA_ChannelPriority1
Channel priority 1

enumerator kDMA_ChannelPriority2
Channel priority 2

enumerator kDMA_ChannelPriority3
Channel priority 3

enumerator kDMA_ChannelPriority4
Channel priority 4

enumerator kDMA_ChannelPriority5
Channel priority 5

enumerator kDMA_ChannelPriority6
Channel priority 6

enumerator kDMA_ChannelPriority7
Lowest channel priority - priority 7

enum _dma_int

DMA interrupt flags.

Values:

enumerator kDMA_IntA
DMA interrupt flag A

enumerator kDMA_IntB
DMA interrupt flag B

enumerator kDMA_IntError
DMA interrupt flag error

enum _dma_trigger_type

DMA trigger type.

Values:

enumerator kDMA_NoTrigger
Trigger is disabled

enumerator kDMA_LowLevelTrigger
Low level active trigger

enumerator kDMA_HighLevelTrigger
High level active trigger

enumerator kDMA_FallingEdgeTrigger
Falling edge active trigger

enumerator kDMA_RisingEdgeTrigger
Rising edge active trigger

_dma_burst_size DMA burst size

Values:

enumerator kDMA_BurstSize1
burst size 1 transfer

enumerator kDMA_BurstSize2
burst size 2 transfer

enumerator kDMA_BurstSize4
burst size 4 transfer

enumerator kDMA_BurstSize8
burst size 8 transfer

enumerator kDMA_BurstSize16
burst size 16 transfer

enumerator kDMA_BurstSize32
burst size 32 transfer

enumerator kDMA_BurstSize64
burst size 64 transfer

enumerator kDMA_BurstSize128
burst size 128 transfer

enumerator kDMA_BurstSize256
burst size 256 transfer

enumerator kDMA_BurstSize512
burst size 512 transfer

enumerator kDMA_BurstSize1024
burst size 1024 transfer

enum _dma_trigger_burst
DMA trigger burst.
Values:

enumerator kDMA_SingleTransfer
Single transfer

enumerator kDMA_LevelBurstTransfer
Burst transfer driven by level trigger

enumerator kDMA_EdgeBurstTransfer1
Perform 1 transfer by edge trigger

enumerator kDMA_EdgeBurstTransfer2
Perform 2 transfers by edge trigger

enumerator kDMA_EdgeBurstTransfer4
Perform 4 transfers by edge trigger

enumerator kDMA_EdgeBurstTransfer8
Perform 8 transfers by edge trigger

enumerator kDMA_EdgeBurstTransfer16
Perform 16 transfers by edge trigger

enumerator kDMA_EdgeBurstTransfer32
Perform 32 transfers by edge trigger

enumerator kDMA_EdgeBurstTransfer64
Perform 64 transfers by edge trigger

enumerator kDMA_EdgeBurstTransfer128
Perform 128 transfers by edge trigger

enumerator kDMA_EdgeBurstTransfer256
Perform 256 transfers by edge trigger

enumerator kDMA_EdgeBurstTransfer512
Perform 512 transfers by edge trigger

enumerator kDMA_EdgeBurstTransfer1024
Perform 1024 transfers by edge trigger

enum _dma_burst_wrap
DMA burst wrapping.
Values:

enumerator kDMA_NoWrap
Wrapping is disabled

enumerator kDMA_SrcWrap
Wrapping is enabled for source

enumerator kDMA_DstWrap
Wrapping is enabled for destination

enumerator kDMA_SrcAndDstWrap
Wrapping is enabled for source and destination

enum _dma_transfer_type
DMA transfer type.
Values:

enumerator kDMA_MemoryToMemory
Transfer from memory to memory (increment source and destination)

enumerator kDMA_PeripheralToMemory
Transfer from peripheral to memory (increment only destination)

enumerator kDMA_MemoryToPeripheral
Transfer from memory to peripheral (increment only source)

enumerator kDMA_StaticToStatic
Peripheral to static memory (do not increment source or destination)

typedef struct *_dma_descriptor* dma_descriptor_t

DMA descriptor structure.

typedef struct *_dma_xfercfg* dma_xfercfg_t

DMA transfer configuration.

typedef enum *_dma_priority* dma_priority_t

DMA channel priority.

typedef enum *_dma_int* dma_irq_t

DMA interrupt flags.

typedef enum *_dma_trigger_type* dma_trigger_type_t

DMA trigger type.

typedef enum *_dma_trigger_burst* dma_trigger_burst_t

DMA trigger burst.

typedef enum *_dma_burst_wrap* dma_burst_wrap_t

DMA burst wrapping.

typedef enum *_dma_transfer_type* dma_transfer_type_t

DMA transfer type.

typedef struct *_dma_channel_trigger* dma_channel_trigger_t

DMA channel trigger.

typedef struct *_dma_channel_config* dma_channel_config_t

DMA channel trigger.

typedef struct *_dma_transfer_config* dma_transfer_config_t

DMA transfer configuration.

typedef void (*dma_callback)(struct *_dma_handle* *handle, void *userData, bool transferDone, uint32_t intmode)

Define Callback function for DMA.

typedef struct *_dma_handle* dma_handle_t

DMA transfer handle structure.

DMA_MAX_TRANSFER_COUNT

DMA max transfer size.

FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE

DMA channel numbers.

DMA head link descriptor table align size

DMA_ALLOCATE_HEAD_DESCRIPTORS(name, number)

DMA head descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.

Parameters

- name – Allocate decriptor name.
- number – Number of descriptor to be allocated.

DMA_ALLOCATE_HEAD_DESCRIPTORS_AT_NONCACHEABLE(name, number)

DMA head descriptor table allocate macro at noncacheable section To simplify user interface, this macro will help allocate descriptor memory at noncacheable section, user just need to provide the name and the number for the allocate descriptor.

Parameters

- name – Allocate decriptor name.
- number – Number of descriptor to be allocated.

DMA_ALLOCATE_LINK_DESCRIPTORS(name, number)

DMA link descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.

Parameters

- name – Allocate decriptor name.
- number – Number of descriptor to be allocated.

DMA_ALLOCATE_LINK_DESCRIPTORS_AT_NONCACHEABLE(name, number)

DMA link descriptor table allocate macro at noncacheable section To simplify user interface, this macro will help allocate descriptor memory at noncacheable section, user just need to provide the name and the number for the allocate descriptor.

Parameters

- name – Allocate decriptor name.
- number – Number of descriptor to be allocated.

DMA_ALLOCATE_DATA_TRANSFER_BUFFER(name, width)

DMA transfer buffer address need to align with the transfer width.

DMA_CHANNEL_GROUP(channel)

DMA_CHANNEL_INDEX(base, channel)

DMA_COMMON_REG_GET(base, channel, reg)

DMA linked descriptor address algin size.

DMA_COMMON_CONST_REG_GET(base, channel, reg)

DMA_COMMON_REG_SET(base, channel, reg, value)

DMA_DESCRIPTOR_END_ADDRESS(start, inc, bytes, width)

DMA descriptor end address calculate.

Parameters

- start – start address
- inc – address interleave size
- bytes – transfer bytes

- width – transfer width

DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc, bytes)

struct _dma_descriptor

#include <fsl_dma.h> DMA descriptor structure.

Public Members

volatile uint32_t xfercfg

Transfer configuration

void *srcEndAddr

Last source address of DMA transfer

void *dstEndAddr

Last destination address of DMA transfer

void *linkToNextDesc

Address of next DMA descriptor in chain

struct _dma_xfercfg

#include <fsl_dma.h> DMA transfer configuration.

Public Members

bool valid

Descriptor is ready to transfer

bool reload

Reload channel configuration register after current descriptor is exhausted

bool swtrig

Perform software trigger. Transfer if fired when 'valid' is set

bool clrtrig

Clear trigger

bool intA

Raises IRQ when transfer is done and set IRQA status register flag

bool intB

Raises IRQ when transfer is done and set IRQB status register flag

uint8_t byteWidth

Byte width of data to transfer

uint8_t srcInc
Increment source address by 'srcInc' x 'byteWidth'

uint8_t dstInc
Increment destination address by 'dstInc' x 'byteWidth'

uint16_t transferCount
Number of transfers

struct __dma__channel__trigger
#include <fsl_dma.h> DMA channel trigger.

Public Members

dma_trigger_type_t type
Select hardware trigger as edge triggered or level triggered.

dma_trigger_burst_t burst
Select whether hardware triggers cause a single or burst transfer.

dma_burst_wrap_t wrap
Select wrap type, source wrap or dest wrap, or both.

struct __dma__channel__config
#include <fsl_dma.h> DMA channel trigger.

Public Members

void *srcStartAddr
Source data address

void *dstStartAddr
Destination data address

void *nextDesc
Chain custom descriptor

uint32_t xferCfg
channel transfer configurations

dma_channel_trigger_t *trigger
DMA trigger type

bool isPeriph
select the request type

struct __dma__transfer__config
#include <fsl_dma.h> DMA transfer configuration.

Public Members

uint8_t *srcAddr
Source data address

uint8_t *dstAddr
Destination data address

uint8_t *nextDesc
Chain custom descriptor

dma_xfercfg_t xfercfg
Transfer options

bool isPeriph
DMA transfer is driven by peripheral

struct __dma__handle
#include <fsl_dma.h> DMA transfer handle structure.

Public Members

dma_callback callback
Callback function. Invoked when transfer of descriptor with interrupt flag finishes

void *userData
Callback function parameter

DMA_Type *base
DMA peripheral base address

uint8_t channel
DMA channel number

2.10 DMIC: Digital Microphone

2.11 DMIC DMA Driver

status_t DMIC_TransferCreateHandleDMA(DMIC_Type *base, *dmic_dma_handle_t* *handle, *dmic_dma_transfer_callback_t* callback, void *userData, *dma_handle_t* *rxDmaHandle)

Initializes the DMIC handle which is used in transactional functions.

Parameters

- base – DMIC peripheral base address.
- handle – Pointer to dmic_dma_handle_t structure.

- callback – Callback function.
- userData – User data.
- rxDmaHandle – User-requested DMA handle for RX DMA transfer.

status_t DMIC_TransferReceiveDMA(DMIC_Type *base, *dmic_dma_handle_t* *handle, *dmic_transfer_t* *xfer, uint32_t channel)

Receives data using DMA.

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – USART peripheral base address.
- handle – Pointer to usart_dma_handle_t structure.
- xfer – DMIC DMA transfer structure. See *dmic_transfer_t*.
- channel – DMIC start channel number.

Return values

kStatus_Success –

void DMIC_TransferAbortReceiveDMA(DMIC_Type *base, *dmic_dma_handle_t* *handle)

Aborts the received data using DMA.

This function aborts the received data using DMA.

Parameters

- base – DMIC peripheral base address
- handle – Pointer to dmic_dma_handle_t structure

status_t DMIC_TransferGetReceiveCountDMA(DMIC_Type *base, *dmic_dma_handle_t* *handle, uint32_t *count)

Get the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – DMIC peripheral base address.
- handle – DMIC 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 DMIC_InstallDMADescriptorMemory(*dmic_dma_handle_t* *handle, void *linkAddr, size_t linkNum)

Install DMA descriptor memory.

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, it should be called after DMIC_TransferCreateHandleDMA. User should be take care about the address of DMA descriptor pool which required align with 16BYTE at least.

Parameters

- handle – Pointer to DMA channel transfer handle.

- linkAddr – DMA link descriptor address.
- linkNum – DMA link descriptor number.

FSL_DMIC_DMA_DRIVER_VERSION
DMIC DMA driver version 2.4.2.

typedef struct *_dmic_transfer* dmic_transfer_t
DMIC transfer structure.

typedef struct *_dmic_dma_handle* dmic_dma_handle_t

typedef void (*dmic_dma_transfer_callback_t)(DMIC_Type *base, *dmic_dma_handle_t* *handle, *status_t* status, void *userData)
DMIC transfer callback function.

struct _dmic_transfer
#include <fsl_dmic_dma.h> DMIC transfer structure.

Public Members

void *data
The buffer of data to be transfer.

uint8_t dataWidth
DMIC support 16bit/32bit

size_t dataSize
The byte count to be transfer.

uint8_t dataAddrInterleaveSize
destination address interleave size

struct *_dmic_transfer* *linkTransfer
use to support link transfer

struct _dmic_dma_handle
#include <fsl_dmic_dma.h> DMIC DMA handle.

Public Members

DMIC_Type *base
DMIC peripheral base address.

dma_handle_t *rxDmaHandle
The DMA RX channel used.

dmic_dma_transfer_callback_t callback
Callback function.

void *userData
DMIC callback function parameter.

size_t transferSize
Size of the data to receive.

volatile uint8_t state
Internal state of DMIC DMA transfer

uint32_t channel
DMIC channel used.

bool isChannelValid
DMIC channel initialization flag

dma_descriptor_t *desLink
descriptor pool pointer

size_t linkNum
number of descriptor in descriptors pool

2.12 DMIC Driver

uint32_t DMIC_GetInstance(DMIC_Type *base)
Get the DMIC instance from peripheral base address.

Parameters

- base – DMIC peripheral base address.

Returns
DMIC instance.

void DMIC_Init(DMIC_Type *base)
Turns DMIC Clock on.

Parameters

- base – : DMIC base

Returns
Nothing

void DMIC_DeInit(DMIC_Type *base)
Turns DMIC Clock off.

Parameters

- base – : DMIC base

Returns
Nothing

void DMIC_SetOperationMode(DMIC_Type *base, *operation_mode_t* mode)

Set DMIC operating mode.

Deprecated:

Do not use this function. It has been superceded by *DMIC_EnableChannelInterrupt*, *DMIC_EnableChannelDma*.

Parameters

- base – : The base address of DMIC interface
- mode – : DMIC mode

Returns

Nothing

void DMIC_Use2fs(DMIC_Type *base, bool use2fs)

Configure Clock scaling.

Parameters

- base – : The base address of DMIC interface
- use2fs – : clock scaling

Returns

Nothing

void DMIC_CfgChannelDc(DMIC_Type *base, *dmic_channel_t* channel, *dc_removal_t* dc_cut_level, uint32_t post_dc_gain_reduce, bool saturate16bit)

Configure DMIC channel.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel
- dc_cut_level – : dc_removal_t, Cut off Frequency
- post_dc_gain_reduce – : Fine gain adjustment in the form of a number of bits to downshift.
- saturate16bit – : If selects 16-bit saturation.

static inline void DMIC_EnableChannelSignExtend(DMIC_Type *base, *dmic_channel_t* channel, bool enable)

Enbale channel sign extend which allows processing of 24bit audio data on 32bit machines.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel
- enable – : true is enable sign extend, false is disable sign extend

void DMIC_ConfigChannel(DMIC_Type *base, *dmic_channel_t* channel, *stereo_side_t* side, *dmic_channel_config_t* *channel_config)

Configure DMIC channel.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel

- side – : stereo_side_t, choice of left or right
- channel_config – : Channel configuration

Returns
Nothing

void DMIC_EnableChannnel(DMIC_Type *base, uint32_t channelmask)

Enable a particualr channel.

Parameters

- base – : The base address of DMIC interface
- channelmask – reference _dmic_channel_mask

Returns
Nothing

void DMIC_FifoChannel(DMIC_Type *base, uint32_t channel, uint32_t trig_level, uint32_t enable, uint32_t resetn)

Configure fifo settings for DMIC channel.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel
- trig_level – : FIFO trigger level
- enable – : FIFO level
- resetn – : FIFO reset

Returns
Nothing

static inline void DMIC_EnableChannelInterrupt(DMIC_Type *base, *dmic_channel_t* channel, bool enable)

Enable a particualr channel interrupt request.

Parameters

- base – : The base address of DMIC interface
- channel – : Channel selection
- enable – : true is enable, false is disable

static inline void DMIC_EnableChannelDma(DMIC_Type *base, *dmic_channel_t* channel, bool enable)

Enable a particualr channel dma request.

Parameters

- base – : The base address of DMIC interface
- channel – : Channel selection
- enable – : true is enable, false is disable

static inline void DMIC_EnableChannelFifo(DMIC_Type *base, *dmic_channel_t* channel, bool enable)

Enable a particualr channel fifo.

Parameters

- base – : The base address of DMIC interface
- channel – : Channel selection

- enable – : true is enable, false is disable

static inline void DMIC_DoFifoReset(DMIC_Type *base, *dmic_channel_t* channel)

Channel fifo reset.

Parameters

- base – : The base address of DMIC interface
- channel – : Channel selection

static inline uint32_t DMIC_FifoGetStatus(DMIC_Type *base, uint32_t channel)

Get FIFO status.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel

Returns

FIFO status

static inline void DMIC_FifoClearStatus(DMIC_Type *base, uint32_t channel, uint32_t mask)

Clear FIFO status.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel
- mask – : Bits to be cleared

Returns

FIFO status

static inline uint32_t DMIC_FifoGetData(DMIC_Type *base, uint32_t channel)

Get FIFO data.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel

Returns

FIFO data

static inline uint32_t DMIC_FifoGetAddress(DMIC_Type *base, uint32_t channel)

Get FIFO address.

Parameters

- base – : The base address of DMIC interface
- channel – : DMIC channel

Returns

FIFO data

void DMIC_ResetChannelDecimator(DMIC_Type *base, uint32_t channelMask, bool reset)

DMIC channel Decimator reset.

Parameters

- base – : The base address of DMIC interface
- channelMask – : DMIC channel mask, reference _dmic_channel_mask
- reset – : true is reset decimator, false is release decimator.

static inline void DMIC_EnableChannelGlobalSync(DMIC_Type *base, uint32_t channelMask, uint32_t syncCounter)

Enable DMIC channel global sync function.

Parameters

- base – : The base address of DMIC interface
- channelMask – : DMIC channel mask, reference _dmic_channel_mask
- syncCounter – :sync counter will trigger a pulse whenever count reaches CCOUNTVAL. If CCOUNTVAL is set to 0, there will be a pulse on every cycle

static inline void DMIC_DisableChannelGlobalSync(DMIC_Type *base, uint32_t channelMask)

Disbale DMIC channel global sync function.

Parameters

- base – : The base address of DMIC interface
- channelMask – : DMIC channel mask, reference _dmic_channel_mask

void DMIC_EnableIntCallback(DMIC_Type *base, *dmic_callback_t* cb)

Enable callback.

This function enables the interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

Parameters

- base – Base address of the DMIC peripheral.
- cb – callback Pointer to store callback function.

Return values

None. –

void DMIC_DisableIntCallback(DMIC_Type *base, *dmic_callback_t* cb)

Disable callback.

This function disables the interrupt for the selected DMIC peripheral.

Parameters

- base – Base address of the DMIC peripheral.
- cb – callback Pointer to store callback function..

Return values

None. –

static inline void DMIC_SetGainNoiseEstHwvad(DMIC_Type *base, uint32_t value)

Sets the gain value for the noise estimator.

Parameters

- base – DMIC base pointer
- value – gain value for the noise estimator.

Return values

None. –

static inline void DMIC_SetGainSignalEstHwvad(DMIC_Type *base, uint32_t value)

Sets the gain value for the signal estimator.

Parameters

- base – DMIC base pointer
- value – gain value for the signal estimator.

Return values
None. –

static inline void DMIC_SetFilterCtrlHwvad(DMIC_Type *base, uint32_t value)

Sets the hwvad filter cutoff frequency parameter.

Parameters

- base – DMIC base pointer
- value – cut off frequency value.

Return values
None. –

static inline void DMIC_SetInputGainHwvad(DMIC_Type *base, uint32_t value)

Sets the input gain of hwvad.

Parameters

- base – DMIC base pointer
- value – input gain value for hwvad.

Return values
None. –

static inline void DMIC_CtrlClrIntrHwvad(DMIC_Type *base, bool st10)

Clears hwvad internal interrupt flag.

Parameters

- base – DMIC base pointer
- st10 – bit value.

Return values
None. –

static inline void DMIC_FilterResetHwvad(DMIC_Type *base, bool rstt)

Resets hwvad filters.

Parameters

- base – DMIC base pointer
- rstt – Reset bit value.

Return values
None. –

static inline uint16_t DMIC_GetNoiseEnvlpEst(DMIC_Type *base)

Gets the value from output of the filter z7.

Parameters

- base – DMIC base pointer

Return values
output – of filter z7.

void DMIC_HwvadEnableIntCallback(DMIC_Type *base, *dmic_hwvad_callback_t* vadcb)

Enable hwvad callback.

This function enables the hwvad interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

Parameters

- base – Base address of the DMIC peripheral.

- vadcb – callback Pointer to store callback function.

Return values
None. –

void DMIC_HwvadDisableIntCallback(DMIC_Type *base, *dmic_hwvad_callback_t* vadcb)

Disable callback.

This function disables the hwvad interrupt for the selected DMIC peripheral.

Parameters

- base – Base address of the DMIC peripheral.
- vadcb – callback Pointer to store callback function..

Return values
None. –

FSL_DMIC_DRIVER_VERSION

DMIC driver version 2.3.4.

_dmic_status DMIC transfer status.

Values:

enumerator kStatus_DMIC_Busy
DMIC is busy

enumerator kStatus_DMIC_Idle
DMIC is idle

enumerator kStatus_DMIC_OverRunError
DMIC over run Error

enumerator kStatus_DMIC_UnderRunError
DMIC under run Error

enum _operation_mode

DMIC different operation modes.

Values:

enumerator kDMIC_OperationModeInterrupt
Interrupt mode

enumerator kDMIC_OperationModeDma
DMA mode

enum _stereo_side

DMIC left/right values.

Values:

enumerator kDMIC_Left
Left Stereo channel

enumerator kDMIC_Right
Right Stereo channel

enum pdm_div_t
DMIC Clock pre-divider values.
Values:

enumerator kDMIC_PdmDiv1
DMIC pre-divider set in divide by 1

enumerator kDMIC_PdmDiv2
DMIC pre-divider set in divide by 2

enumerator kDMIC_PdmDiv3
DMIC pre-divider set in divide by 3

enumerator kDMIC_PdmDiv4
DMIC pre-divider set in divide by 4

enumerator kDMIC_PdmDiv6
DMIC pre-divider set in divide by 6

enumerator kDMIC_PdmDiv8
DMIC pre-divider set in divide by 8

enumerator kDMIC_PdmDiv12
DMIC pre-divider set in divide by 12

enumerator kDMIC_PdmDiv16
DMIC pre-divider set in divide by 16

enumerator kDMIC_PdmDiv24
DMIC pre-divider set in divide by 24

enumerator kDMIC_PdmDiv32
DMIC pre-divider set in divide by 32

enumerator kDMIC_PdmDiv48
DMIC pre-divider set in divide by 48

enumerator kDMIC_PdmDiv64
DMIC pre-divider set in divide by 64

enumerator kDMIC_PdmDiv96
DMIC pre-divider set in divide by 96

enumerator kDMIC_PdmDiv128
DMIC pre-divider set in divide by 128

enum _compensation

Pre-emphasis Filter coefficient value for 2FS and 4FS modes.

Values:

enumerator kDMIC_CompValueZero
Compensation 0

enumerator kDMIC_CompValueNegativePoint16
Compensation -0.16

enumerator kDMIC_CompValueNegativePoint15
Compensation -0.15

enumerator kDMIC_CompValueNegativePoint13
Compensation -0.13

enum _dc_removal

DMIC DC filter control values.

Values:

enumerator kDMIC_DcNoRemove
Flat response no filter

enumerator kDMIC_DcCut155
Cut off Frequency is 155 Hz

enumerator kDMIC_DcCut78
Cut off Frequency is 78 Hz

enumerator kDMIC_DcCut39
Cut off Frequency is 39 Hz

enum _dmic_channel

DMIC Channel number.

Values:

enumerator kDMIC_Channel0
DMIC channel 0

enumerator kDMIC_Channel1
DMIC channel 1

enumerator kDMIC_ChannelMAX
Maximum number of DMIC channels

_dmic_channel_mask DMIC Channel mask.

Values:

enumerator kDMIC_EnableChannel0
DMIC channel 0 mask

enumerator kDMIC_EnableChannel1
DMIC channel 1 mask

enum _dmic_phy_sample_rate
DMIC and decimator sample rates.
Values:

enumerator kDMIC_PhyFullSpeed
Decimator gets one sample per each chosen clock edge of PDM interface

enumerator kDMIC_PhyHalfSpeed
PDM clock to Microphone is halved, decimator receives each sample twice

typedef enum *_operation_mode* operation_mode_t
DMIC different operation modes.

typedef enum *_stereo_side* stereo_side_t
DMIC left/right values.

typedef enum *_compensation* compensation_t
Pre-emphasis Filter coefficient value for 2FS and 4FS modes.

typedef enum *_dc_removal* dc_removal_t
DMIC DC filter control values.

typedef enum *_dmic_channel* dmic_channel_t
DMIC Channel number.

typedef enum *_dmic_phy_sample_rate* dmic_phy_sample_rate_t
DMIC and decimator sample rates.

typedef struct *_dmic_channel_config* dmic_channel_config_t
DMIC Channel configuration structure.

typedef void (*dmic_callback_t)(void)
DMIC Callback function.

typedef void (*dmic_hwvad_callback_t)(void)
HWVAD Callback function.

struct _dmic_channel_config
#include <fsl_dmic.h> DMIC Channel configuration structure.

Public Members

pdm_div_t divhfclk
DMIC Clock pre-divider values

uint32_t osr
oversampling rate(CIC decimation rate) for PCM

uint32_t gainshft
4FS PCM data gain control

compensation_t preac2coef
Pre-emphasis Filter coefficient value for 2FS

compensation_t preac4coef
Pre-emphasis Filter coefficient value for 4FS

dc_removal_t dc_cut_level
DMIC DC filter control values.

uint32_t post_dc_gain_reduce
Fine gain adjustment in the form of a number of bits to downshift

dmic_phy_sample_rate_t sample_rate
DMIC and decimator sample rates

bool saturate16bit
Selects 16-bit saturation. 0 means results roll over if out range and do not saturate. 1 means if the result overflows, it saturates at 0xFFFF for positive overflow and 0x8000 for negative overflow.

bool enableSignExtend
sign extend feature which allows processing of 24bit audio data on 32bit machine

2.13 DSP Driver

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.

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.

FSL_DSP_DRIVER_VERSION

reset driver version 2.1.1.

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.

2.14 FLEXCOMM: FLEXCOMM Driver

2.15 FLEXCOMM Driver

FSL_FLEXCOMM_DRIVER_VERSION

FlexCOMM driver version 2.0.3.

enum FLEXCOMM_PERIPH_T

FLEXCOMM peripheral modes.

Values:

enumerator FLEXCOMM_PERIPH_NONE

No peripheral

enumerator FLEXCOMM_PERIPH_USART

USART peripheral

enumerator FLEXCOMM_PERIPH_SPI

SPI Peripheral

enumerator FLEXCOMM_PERIPH_I2C
I2C Peripheral

enumerator FLEXCOMM_PERIPH_I2S_TX
I2S TX Peripheral

enumerator FLEXCOMM_PERIPH_I2S_RX
I2S RX Peripheral

typedef void (*flexcomm_irq_handler_t)(void *base, void *handle)
Typedef for interrupt handler.

IRQn_Type const kFlexcommIrqs[]
Array with IRQ number for each FLEXCOMM module.

uint32_t FLEXCOMM_GetInstance(void *base)
Returns instance number for FLEXCOMM module with given base address.

status_t FLEXCOMM_Init(void *base, *FLEXCOMM_PERIPH_T* periph)
Initializes FLEXCOMM and selects peripheral mode according to the second parameter.

void FLEXCOMM_SetIRQHandler(void *base, *flexcomm_irq_handler_t* handler, void *flexcommHandle)
Sets IRQ handler for given FLEXCOMM module. It is used by drivers register IRQ handler according to FLEXCOMM mode.

void FLEXCOMM_DriverIRQHandler(uint32_t instance)
Common IRQ handler for all FLEXCOMM instances.

2.16 FLEXSPI: Flexible Serial Peripheral Interface Driver

uint32_t FLEXSPI_GetInstance(FLEXSPI_Type *base)
Get the instance number for FLEXSPI.

Parameters

- base – FLEXSPI base pointer.

status_t FLEXSPI_CheckAndClearError(FLEXSPI_Type *base, uint32_t status)
Check and clear IP command execution errors.

Parameters

- base – FLEXSPI base pointer.
- status – interrupt status.

void FLEXSPI_Init(FLEXSPI_Type *base, const *flexspi_config_t* *config)
Initializes the FLEXSPI module and internal state.

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

- base – FLEXSPI peripheral base address.
- config – FLEXSPI configure structure.

void FLEXSPI_GetDefaultConfig(*flexspi_config_t* *config)

Gets default settings for FLEXSPI.

Parameters

- config – FLEXSPI configuration structure.

void FLEXSPI_Deinit(FLEXSPI_Type *base)

Deinitializes the FLEXSPI module.

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

- base – FLEXSPI peripheral base address.

void FLEXSPI_UpdateDllValue(FLEXSPI_Type *base, *flexspi_device_config_t* *config, *flexspi_port_t* port)

Update FLEXSPI DLL value depending on currently flexspi root clock.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI_SetFlashConfig(FLEXSPI_Type *base, *flexspi_device_config_t* *config, *flexspi_port_t* port)

Configures the connected device parameter.

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

Parameters

- base – FLEXSPI peripheral base address.
- config – Flash configuration parameters.
- port – FLEXSPI Operation port.

void FLEXSPI_SoftwareReset(FLEXSPI_Type *base)

Software reset for the FLEXSPI logic.

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

- base – FLEXSPI peripheral base address.

static inline void FLEXSPI_Enable(FLEXSPI_Type *base, bool enable)

Enables or disables the FLEXSPI module.

Parameters

- base – FLEXSPI peripheral base address.
- enable – True means enable FLEXSPI, false means disable.

void FLEXSPI_UpdateAhbBuffersSettings(FLEXSPI_Type *base, *flexspi_ahbBuffers_ctrl_t* *ptrAhbBufferCtrl)

Update all AHB buffers' settings, including buffer size, master ID.

Parameters

- base – FLEXSPI peripheral base address.

- ptrAhbBufferCtrl – Pointer to structure *flexspi_ahbBuffers_ctrl_t* which store all AHB buffers' settings.

static inline void FLEXSPI_EnableInterrupts(FLEXSPI_Type *base, uint32_t mask)

Enables the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline void FLEXSPI_DisableInterrupts(FLEXSPI_Type *base, uint32_t mask)

Disable the FLEXSPI interrupts.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline void FLEXSPI_EnableTxDMA(FLEXSPI_Type *base, bool enable)

Enables or disables FLEXSPI IP Tx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for transmit DMA request. Pass true for enable, false for disable.

static inline void FLEXSPI_EnableRxDMA(FLEXSPI_Type *base, bool enable)

Enables or disables FLEXSPI IP Rx FIFO DMA requests.

Parameters

- base – FLEXSPI peripheral base address.
- enable – Enable flag for receive DMA request. Pass true for enable, false for disable.

static inline uint32_t FLEXSPI_GetTxFifoAddress(FLEXSPI_Type *base)

Gets FLEXSPI IP tx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – tx fifo address.

static inline uint32_t FLEXSPI_GetRxFifoAddress(FLEXSPI_Type *base)

Gets FLEXSPI IP rx fifo address for DMA transfer.

Parameters

- base – FLEXSPI peripheral base address.

Return values

The – rx fifo address.

static inline void FLEXSPI_ResetFifos(FLEXSPI_Type *base, bool txFifo, bool rxFifo)

Clears the FLEXSPI IP FIFO logic.

Parameters

- base – FLEXSPI peripheral base address.
- txFifo – Pass true to reset TX FIFO.
- rxFifo – Pass true to reset RX FIFO.

static inline void FLEXSPI_GetFifoCounts(FLEXSPI_Type *base, size_t *txCount, size_t *rxCount)

Gets the valid data entries in the FLEXSPI FIFOs.

Parameters

- base – FLEXSPI 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 uint32_t FLEXSPI_GetInterruptStatusFlags(FLEXSPI_Type *base)

Get the FLEXSPI interrupt status flags.

Parameters

- base – FLEXSPI peripheral base address.

Return values

interrupt – status flag, use status flag to AND *flexspi_flags_t* could get the related status.

static inline void FLEXSPI_ClearInterruptStatusFlags(FLEXSPI_Type *base, uint32_t mask)

Get the FLEXSPI interrupt status flags.

Parameters

- base – FLEXSPI peripheral base address.
- mask – FLEXSPI interrupt source.

static inline *flexspi_arb_command_source_t* FLEXSPI_GetArbitratorCommandSource(FLEXSPI_Type *base)

Gets the trigger source of current command sequence granted by arbitrator.

Parameters

- base – FLEXSPI peripheral base address.

Return values

trigger – source of current command sequence.

static inline *flexspi_ip_error_code_t* FLEXSPI_GetIPCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)

Gets the error code when IP command error detected.

Parameters

- base – FLEXSPI peripheral base address.
- index – Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values

error – code when IP command error detected.

static inline *flexspi_ahb_error_code_t* FLEXSPI_GetAHBCommandErrorCode(FLEXSPI_Type *base, uint8_t *index)

Gets the error code when AHB command error detected.

Parameters

- base – FLEXSPI peripheral base address.

- index – Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values

error – code when AHB command error detected.

static inline bool FLEXSPI_GetBusIdleStatus(FLEXSPI_Type *base)

Returns whether the bus is idle.

Parameters

- base – FLEXSPI peripheral base address.

Return values

- true – Bus is idle.
- false – Bus is busy.

void FLEXSPI_UpdateRxSampleClock(FLEXSPI_Type *base, *flexspi_read_sample_clock_t* clockSource)

Update read sample clock source.

Parameters

- base – FLEXSPI peripheral base address.
- clockSource – clockSource of type *flexspi_read_sample_clock_t*

void FLEXSPI_UpdateLUT(FLEXSPI_Type *base, uint32_t index, const uint32_t *cmd, uint32_t count)

Updates the LUT table.

Parameters

- base – FLEXSPI 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 8 instructions and occupy 4*32-bit memory.
- cmd – Command sequence array.
- count – Number of sequences.

static inline void FLEXSPI_WriteData(FLEXSPI_Type *base, uint32_t data, uint8_t fifoIndex)

Writes data into FIFO.

Parameters

- base – FLEXSPI peripheral base address
- data – The data bytes to send
- fifoIndex – Destination fifo index.

static inline uint32_t FLEXSPI_ReadData(FLEXSPI_Type *base, uint8_t fifoIndex)

Receives data from data FIFO.

Parameters

- base – FLEXSPI peripheral base address
- fifoIndex – Source fifo index.

Returns

The data in the FIFO.

status_t FLEXSPI_WriteBlocking(FLEXSPI_Type *base, uint8_t *buffer, size_t size)

Sends a buffer of data bytes using blocking method.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – FLEXSPI peripheral base address
- buffer – The data bytes to send
- size – The number of data bytes to send

Return values

- kStatus_Success – write success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t FLEXSPI_ReadBlocking(FLEXSPI_Type *base, uint8_t *buffer, size_t size)

Receives a buffer of data bytes using a blocking method.

Note: This function blocks via polling until all bytes have been sent.

Parameters

- base – FLEXSPI peripheral base address
- buffer – The data bytes to send
- size – The number of data bytes to receive

Return values

- kStatus_Success – read success without error
- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequencen error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

status_t FLEXSPI_TransferBlocking(FLEXSPI_Type *base, *flexspi_transfer_t* *xfer)

Execute command to transfer a buffer data bytes using a blocking method.

Parameters

- base – FLEXSPI peripheral base address
- xfer – pointer to the transfer structure.

Return values

- kStatus_Success – command transfer success without error

- kStatus_FLEXSPI_SequenceExecutionTimeout – sequence execution timeout
- kStatus_FLEXSPI_IpCommandSequenceError – IP command sequence error detected
- kStatus_FLEXSPI_IpCommandGrantTimeout – IP command grant timeout detected

void FLEXSPI_TransferCreateHandle(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *flexspi_transfer_callback_t* callback, void *userData)

Initializes the FLEXSPI handle which is used in transactional functions.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure to store the transfer state.
- callback – pointer to user callback function.
- userData – user parameter passed to the callback function.

status_t FLEXSPI_TransferNonBlocking(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *flexspi_transfer_t* *xfer)

Performs a interrupt non-blocking transfer on the FLEXSPI bus.

Note: Calling the API returns immediately after transfer initiates. The user needs to call FLEXSPI_GetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXSPI_Busy, the transfer is finished. For FLEXSPI_Read, the dataSize should be multiple of rx watermark level, or FLEXSPI could not read data properly.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure which stores the transfer state.
- xfer – pointer to flexspi_transfer_t structure.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_FLEXSPI_Busy – Previous transmission still not finished.

status_t FLEXSPI_TransferGetCount(FLEXSPI_Type *base, *flexspi_handle_t* *handle, size_t *count)

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_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_Success – Successfully return the count.

void FLEXSPI_TransferAbort(FLEXSPI_Type *base, *flexspi_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 – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure which stores the transfer state

void FLEXSPI_TransferHandleIRQ(FLEXSPI_Type *base, *flexspi_handle_t* *handle)

Master interrupt handler.

Parameters

- base – FLEXSPI peripheral base address.
- handle – pointer to flexspi_handle_t structure.

FSL_FLEXSPI_DRIVER_VERSION

FLEXSPI driver version.

Status structure of FLEXSPI.

Values:

enumerator kStatus_FLEXSPI_Busy

FLEXSPI is busy

enumerator kStatus_FLEXSPI_SequenceExecutionTimeout

Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator kStatus_FLEXSPI_IpCommandSequenceError

IP command Sequence execution timeout error occurred during FLEXSPI transfer.

enumerator kStatus_FLEXSPI_IpCommandGrantTimeout

IP command grant timeout error occurred during FLEXSPI transfer.

CMD definition of FLEXSPI, use to form LUT instruction, _flexspi_command.

Values:

enumerator kFLEXSPI_Command_STOP

Stop execution, deassert CS.

enumerator kFLEXSPI_Command_SDR

Transmit Command code to Flash, using SDR mode.

enumerator kFLEXSPI_Command_RADDR_SDR
Transmit Row Address to Flash, using SDR mode.

enumerator kFLEXSPI_Command_CADDR_SDR
Transmit Column Address to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE1_SDR
Transmit 1-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE2_SDR
Transmit 2-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE4_SDR
Transmit 4-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_MODE8_SDR
Transmit 8-bit Mode bits to Flash, using SDR mode.

enumerator kFLEXSPI_Command_WRITE_SDR
Transmit Programming Data to Flash, using SDR mode.

enumerator kFLEXSPI_Command_READ_SDR
Receive Read Data from Flash, using SDR mode.

enumerator kFLEXSPI_Command_LEARN_SDR
Receive Read Data or Preamble bit from Flash, SDR mode.

enumerator kFLEXSPI_Command_DATSZ_SDR
Transmit Read/Program Data size (byte) to Flash, SDR mode.

enumerator kFLEXSPI_Command_DUMMY_SDR
Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI_Command_DUMMY_RWDS_SDR
Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI_Command_DDR
Transmit Command code to Flash, using DDR mode.

enumerator kFLEXSPI_Command_RADDR_DDR
Transmit Row Address to Flash, using DDR mode.

enumerator kFLEXSPI_Command_CADDR_DDR
Transmit Column Address to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE1_DDR
Transmit 1-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE2_DDR
Transmit 2-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE4_DDR
Transmit 4-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_MODE8_DDR
Transmit 8-bit Mode bits to Flash, using DDR mode.

enumerator kFLEXSPI_Command_WRITE_DDR
Transmit Programming Data to Flash, using DDR mode.

enumerator kFLEXSPI_Command_READ_DDR
Receive Read Data from Flash, using DDR mode.

enumerator kFLEXSPI_Command_LEARN_DDR
Receive Read Data or Preamble bit from Flash, DDR mode.

enumerator kFLEXSPI_Command_DATSZ_DDR
Transmit Read/Program Data size (byte) to Flash, DDR mode.

enumerator kFLEXSPI_Command_DUMMY_DDR
Leave data lines undriven by FlexSPI controller.

enumerator kFLEXSPI_Command_DUMMY_RWDS_DDR
Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

enumerator kFLEXSPI_Command_JUMP_ON_CS
Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence

enum _flexspi_pad
pad definition of FLEXSPI, use to form LUT instruction.

Values:

enumerator kFLEXSPI_1PAD
Transmit command/address and transmit/receive data only through DATA0/DATA1.

enumerator kFLEXSPI_2PAD
Transmit command/address and transmit/receive data only through DATA[1:0].

enumerator kFLEXSPI_4PAD
Transmit command/address and transmit/receive data only through DATA[3:0].

enumerator kFLEXSPI_8PAD
Transmit command/address and transmit/receive data only through DATA[7:0].

enum _flexspi_flags

FLEXSPI interrupt status flags.

Values:

enumerator kFLEXSPI_SequenceExecutionTimeoutFlag

Sequence execution timeout.

enumerator kFLEXSPI_AhbBusErrorFlag

AHB Bus error flag.

enumerator kFLEXSPI_SckStoppedBecauseTxEmptyFlag

SCK is stopped during command sequence because Async TX FIFO empty.

enumerator kFLEXSPI_SckStoppedBecauseRxFullFlag

SCK is stopped during command sequence because Async RX FIFO full.

enumerator kFLEXSPI_IpTxFifoWatermarkEmptyFlag

IP TX FIFO WaterMark empty.

enumerator kFLEXSPI_IpRxFifoWatermarkAvailableFlag

IP RX FIFO WaterMark available.

enumerator kFLEXSPI_AhbCommandSequenceErrorFlag

AHB triggered Command Sequences Error.

enumerator kFLEXSPI_IpCommandSequenceErrorFlag

IP triggered Command Sequences Error.

enumerator kFLEXSPI_AhbCommandGrantTimeoutFlag

AHB triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandGrantTimeoutFlag

IP triggered Command Sequences Grant Timeout.

enumerator kFLEXSPI_IpCommandExecutionDoneFlag

IP triggered Command Sequences Execution finished.

enumerator kFLEXSPI_AllInterruptFlags

All flags.

enum _flexspi_read_sample_clock

FLEXSPI sample clock source selection for Flash Reading.

Values:

enumerator kFLEXSPI_ReadSampleClkLoopbackInternally

Dummy Read strobe generated by FlexSPI Controller and loopback internally.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromDqsPad
Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

enumerator kFLEXSPI_ReadSampleClkLoopbackFromSckPad
SCK output clock and loopback from SCK pad.

enumerator kFLEXSPI_ReadSampleClkExternalInputFromDqsPad
Flash provided Read strobe and input from DQS pad.

enum _flexspi_cs_interval_cycle_unit
FLEXSPI interval unit for flash device select.
Values:

enumerator kFLEXSPI_CsIntervalUnit1SckCycle
Chip selection interval: CSINTERVAL * 1 serial clock cycle.

enumerator kFLEXSPI_CsIntervalUnit256SckCycle
Chip selection interval: CSINTERVAL * 256 serial clock cycle.

enum _flexspi_ahb_write_wait_unit
FLEXSPI AHB wait interval unit for writing.
Values:

enumerator kFLEXSPI_AhbWriteWaitUnit2AhbCycle
AWRWAIT unit is 2 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8AhbCycle
AWRWAIT unit is 8 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32AhbCycle
AWRWAIT unit is 32 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit128AhbCycle
AWRWAIT unit is 128 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit512AhbCycle
AWRWAIT unit is 512 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit2048AhbCycle
AWRWAIT unit is 2048 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit8192AhbCycle
AWRWAIT unit is 8192 ahb clock cycle.

enumerator kFLEXSPI_AhbWriteWaitUnit32768AhbCycle
AWRWAIT unit is 32768 ahb clock cycle.

enum _flexspi_ip_error_code

Error Code when IP command Error detected.

Values:

enumerator kFLEXSPI_IpCmdErrorNoError

No error.

enumerator kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd

IP command with JMP_ON_CS instruction used.

enumerator kFLEXSPI_IpCmdErrorUnknownOpCode

Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence

Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence

Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_IpCmdErrorInvalidAddress

Flash access start address exceed the whole flash address range (A1/A2/B1/B2).

enumerator kFLEXSPI_IpCmdErrorSequenceExecutionTimeout

Sequence execution timeout.

enumerator kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss

Flash boundary crossed.

enum _flexspi_ahb_error_code

Error Code when AHB command Error detected.

Values:

enumerator kFLEXSPI_AhbCmdErrorNoError

No error.

enumerator kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd

AHB Write command with JMP_ON_CS instruction used in the sequence.

enumerator kFLEXSPI_AhbCmdErrorUnknownOpCode

Unknown instruction opcode in the sequence.

enumerator kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence

Instruction DUMMY_SDR/DUMMY_RWDS_SDR used in DDR sequence.

enumerator kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence

Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.

enumerator kFLEXSPI_AhbCmdSequenceExecutionTimeout
Sequence execution timeout.

enum _flexspi_port
FLEXSPI operation port select.
Values:

enumerator kFLEXSPI_PortA1
Access flash on A1 port.

enumerator kFLEXSPI_PortA2
Access flash on A2 port.

enumerator kFLEXSPI_PortCount

enum _flexspi_arb_command_source
Trigger source of current command sequence granted by arbitrator.
Values:

enumerator kFLEXSPI_AhbReadCommand

enumerator kFLEXSPI_AhbWriteCommand

enumerator kFLEXSPI_IpCommand

enumerator kFLEXSPI_SuspendedCommand

enum _flexspi_command_type
Command type.
Values:

enumerator kFLEXSPI_Command
FlexSPI operation: Only command, both TX and Rx buffer are ignored.

enumerator kFLEXSPI_Config
FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

enumerator kFLEXSPI_Read

enumerator kFLEXSPI_Write

typedef enum *_flexspi_pad* flexspi_pad_t
pad definition of FLEXSPI, use to form LUT instruction.

typedef enum *_flexspi_flags* flexspi_flags_t
FLEXSPI interrupt status flags.

typedef enum *_flexspi_read_sample_clock* flexspi_read_sample_clock_t
FLEXSPI sample clock source selection for Flash Reading.

typedef enum *_flexspi_cs_interval_cycle_unit* flexspi_cs_interval_cycle_unit_t
FLEXSPI interval unit for flash device select.

typedef enum *_flexspi_ahb_write_wait_unit* flexspi_ahb_write_wait_unit_t
FLEXSPI AHB wait interval unit for writing.

typedef enum *_flexspi_ip_error_code* flexspi_ip_error_code_t
Error Code when IP command Error detected.

typedef enum *_flexspi_ahb_error_code* flexspi_ahb_error_code_t
Error Code when AHB command Error detected.

typedef enum *_flexspi_port* flexspi_port_t
FLEXSPI operation port select.

typedef enum *_flexspi_arb_command_source* flexspi_arb_command_source_t
Trigger source of current command sequence granted by arbitrator.

typedef enum *_flexspi_command_type* flexspi_command_type_t
Command type.

typedef struct *_flexspi_ahbBuffer_config* flexspi_ahbBuffer_config_t

typedef struct *_flexspi_ahbBuffers_ctrl* flexspi_ahbBuffers_ctrl_t
Structure to control all AHB buffers.

typedef struct *_flexspi_config* flexspi_config_t
FLEXSPI configuration structure.

typedef struct *_flexspi_device_config* flexspi_device_config_t
External device configuration items.

typedef struct *_flexspi_transfer* flexspi_transfer_t
Transfer structure for FLEXSPI.

typedef struct *_flexspi_handle* flexspi_handle_t

typedef void (*flexspi_transfer_callback_t)(FLEXSPI_Type *base, *flexspi_handle_t* *handle, *status_t* status, void *userData)
FLEXSPI transfer callback function.

typedef struct *_flexspi_addr_map_config* flexspi_addr_map_config_t
Address mapping configuration structure.

FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT

FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)

Formula to form FLEXSPI instructions in LUT table.

struct _flexspi_ahbBuffer_config

#include <fsl_flexspi.h>

Public Members

uint8_t priority

This priority for AHB Master Read which this AHB RX Buffer is assigned.

uint8_t masterIndex

AHB Master ID the AHB RX Buffer is assigned.

uint16_t bufferSize

AHB buffer size in byte.

bool enablePrefetch

AHB Read Prefetch Enable for current AHB RX Buffer corresponding Master, allows prefetch disable/enable separately for each master.

struct _flexspi_ahbBuffers_ctrl

#include <fsl_flexspi.h> Structure to control all AHB buffers.

Public Members

flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]

Configurations of all AHB buffers.

struct _flexspi_config

#include <fsl_flexspi.h> FLEXSPI configuration structure.

Public Members

uint8_t clockDiv

FLEXSPI serial root clock divider.

flexspi_read_sample_clock_t rxSampleClock

Sample Clock source selection for Flash Reading.

bool enableSckFreeRunning

Enable/disable SCK output free-running.

bool enableDoze

Enable/disable doze mode support.

bool enableHalfSpeedAccess

Enable/disable divide by 2 of the clock for half speed commands.

flexspi_read_sample_clock_t rxSampleClockPortB

Sample Clock source_b selection for Flash Reading.

bool rxSampleClockDiff

Sample Clock source or source_b selection for Flash Reading.

bool enableSameConfigForAll

Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.

uint16_t seqTimeoutCycle

Timeout wait cycle for command sequence execution, timeout after ahbGrantTimeoutCyle*1024 serial root clock cycles.

uint8_t ipGrantTimeoutCycle

Timeout wait cycle for IP command grant, timeout after ipGrantTimeoutCycle*1024 AHB clock cycles.

uint8_t txWatermark

FLEXSPI IP transmit watermark value.

uint8_t rxWatermark

FLEXSPI receive watermark value.

struct __flexspi__device__config

#include <fsl_flexspi.h> External device configuration items.

Public Members

uint32_t flexspiRootClk

FLEXSPI serial root clock.

bool isSck2Enabled

FLEXSPI use SCK2.

uint32_t flashSize

Flash size in KByte.

bool addressShift

Address shift.

flexspi_cs_interval_cycle_unit_t CSIntervalUnit

CS interval unit, 1 or 256 cycle.

uint16_t CSInterval
: CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.

uint8_t CSHoldTime
: CS line hold time.

uint8_t CSSetupTime
: CS line setup time.

uint8_t dataValidTime
: Data valid time for external device.

uint8_t columnspace
: Column space size.

bool enableWordAddress
: If enable word address.

uint8_t AWRSeqIndex
: Sequence ID for AHB write command.

uint8_t AWRSeqNumber
: Sequence number for AHB write command.

uint8_t ARDSeqIndex
: Sequence ID for AHB read command.

uint8_t ARDSeqNumber
: Sequence number for AHB read command.

flexspi_ahb_write_wait_unit_t AHBWriteWaitUnit
: AHB write wait unit.

uint16_t AHBWriteWaitInterval
: AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.

bool enableWriteMask
: Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.

bool isFroClockSource
: Is FRO clock source or not.

struct __flexspi__transfer
: *#include <fsl_flexspi.h>* Transfer structure for FLEXSPI.

Public Members

uint32_t deviceAddress
: Operation device address.

flexspi_port_t port
: Operation port.

flexspi_command_type_t cmdType
: Execution command type.

uint8_t seqIndex
: Sequence ID for command.

uint8_t SeqNumber
: Sequence number for command.

uint32_t *data
: Data buffer.

size_t dataSize
: Data size in bytes.

struct __flexspi_handle
: *#include <fsl_flexspi.h>* Transfer handle structure for FLEXSPI.

Public Members

uint32_t state
: Internal state for FLEXSPI transfer

uint8_t *data
: Data buffer.

size_t dataSize
: Remaining Data size in bytes.

size_t transferTotalSize
: Total Data size in bytes.

flexspi_transfer_callback_t completionCallback
: Callback for users while transfer finish or error occurred

void *userData
: FLEXSPI callback function parameter.

struct __flexspi_addr_map_config
: *#include <fsl_flexspi.h>* Address mapping configuration structure.

Public Members

uint32_t addrStart

Remapping start address.

uint32_t addrEnd

Remapping end address.

uint32_t addrOffset

Address offset.

bool remapEnable

Enable address remapping.

struct ahbConfig

Public Members

uint8_t ahbGrantTimeoutCycle

Timeout wait cycle for AHB command grant, timeout after ahbGrantTimeoutCyle*1024 AHB clock cycles.

uint16_t ahbBusTimeoutCycle

Timeout wait cycle for AHB read/write access, timeout after ahbBusTimeoutCycle*1024 AHB clock cycles.

uint8_t resumeWaitCycle

Wait cycle for idle state before suspended command sequence resume, timeout after ahbBusTimeoutCycle AHB clock cycles.

bool disableAhbReadResume

True: Suspended AHB read prefetch does not resume once aborted; **False:** Suspended AHB read prefetch resumes when AHB is IDLE.

flexspi_ahbBuffer_config_t buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNTn(0)]

AHB buffer size.

bool enableClearAHBBufferOpt

Enable/disable automatically clean AHB RX Buffer and TX Buffer when FLEXSPI returns STOP mode ACK.

bool enableReadAddressOpt

Enable/disable remove AHB read burst start address alignment limitation. when enable, there is no AHB read burst start address alignment limitation.

bool enableAHBPrefetch

Enable/disable AHB read prefetch feature, when enabled, FLEXSPI will fetch more data than current AHB burst.

bool enableAHBBufferable

Enable/disable AHB bufferable write access support, when enabled, FLEXSPI return before waiting for command execution finished.

bool enableAHBCachable

Enable AHB bus cachable read access support.

2.17 FLEXSPI DMA Driver

void FLEXSPI_TransferCreateHandleDMA(FLEXSPI_Type *base, *flexspi_dma_handle_t* *handle, *flexspi_dma_callback_t* callback, void *userData, *dma_handle_t* *txDmaHandle, *dma_handle_t* *rxDmaHandle)

Initializes the FLEXSPI handle for transfer which is used in transactional functions and set the callback.

Parameters

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi_dma_handle_t structure
- callback – FLEXSPI 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 FLEXSPI_TransferUpdateSizeDMA(FLEXSPI_Type *base, *flexspi_dma_handle_t* *handle, *flexspi_dma_transfer_nsize_t* nsize)

Update FLEXSPI DMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).

See also:

flexspi_dma_transfer_nsize_t .

Parameters

- base – FLEXSPI peripheral base address
- handle – Pointer to flexspi_dma_handle_t structure
- nsize – FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is *kFLEXPSI_DMAnSize1Bytes(one byte)*.

status_t FLEXSPI_TransferDMA(FLEXSPI_Type *base, *flexspi_dma_handle_t* *handle, *flexspi_transfer_t* *xfer)

Transfers FLEXSPI data using an dma non-blocking method.

This function writes/receives data to/from the FLEXSPI transmit/receive FIFO. This function is non-blocking.

Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi_dma_handle_t structure

- xfer – FLEXSPI transfer structure.

Return values

- kStatus_FLEXSPI_Busy – FLEXSPI is busy transfer.
- kStatus_InvalidArgument – The watermark configuration is invalid, the watermark should be power of 2 to do successfully DMA transfer.
- kStatus_Success – FLEXSPI successfully start dma transfer.

void FLEXSPI_TransferAbortDMA(FLEXSPI_Type *base, *flexspi_dma_handle_t* *handle)

Aborts the transfer data using dma.

This function aborts the transfer data using dma.

Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi_dma_handle_t structure

status_t FLEXSPI_TransferGetTransferCountDMA(FLEXSPI_Type *base, *flexspi_dma_handle_t* *handle, size_t *count)

Gets the transferred counts of transfer.

Parameters

- base – FLEXSPI peripheral base address.
- handle – Pointer to flexspi_dma_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.

FSL_FLEXSPI_DMA_DRIVER_VERSION

FLEXSPI DMA driver version 2.2.3.

enum _flexspi_dma_ntransfer_size

dma transfer configuration

Values:

enumerator kFLEXPSI_DMAnSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kFLEXPSI_DMAnSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kFLEXPSI_DMAnSize4Bytes

Source/Destination data transfer size is 4 bytes every time

typedef struct *_flexspi_dma_handle* flexspi_dma_handle_t

typedef void (*flexspi_dma_callback_t)(FLEXSPI_Type *base, *flexspi_dma_handle_t* *handle, *status_t* status, void *userData)

FLEXSPI dma transfer callback function for finish and error.

typedef enum *_flexspi_dma_ntransfer_size* flexspi_dma_transfer_nsize_t

dma transfer configuration

struct _flexspi_dma_handle

#include <fsl_flexspi_dma.h> FLEXSPI DMA transfer handle, users should not touch the content of the handle.

Public Members

dma_handle_t *txDmaHandle

dma handler for FLEXSPI Tx.

dma_handle_t *rxDmaHandle

dma handler for FLEXSPI Rx.

size_t transferSize

Bytes need to transfer.

flexspi_dma_transfer_nsize_t nsize

dma SSIZE/DSIZE in each transfer.

uint8_t nbytes

dma minor byte transfer count initially configured.

uint8_t count

The transfer data count in a DMA request.

uint32_t state

Internal state for FLEXSPI dma transfer.

flexspi_dma_callback_t completionCallback

A callback function called after the dma transfer is finished.

void *userData

User callback parameter

2.18 FMEAS: Frequency Measure Driver

static inline void FMEAS_StartMeasure(*FMEAS_SYSCON_Type* *base)

Starts a frequency measurement cycle.

Parameters

- base – : SYSCON peripheral base address.

static inline bool FMEAS_IsMeasureComplete(*FMEAS_SYSCON_Type* *base)

Indicates when a frequency measurement cycle is complete.

Parameters

- base – : SYSCON peripheral base address.

Returns
true if a measurement cycle is active, otherwise false.

uint32_t FMEAS_GetFrequency(*FMEAS_SYSCON_Type* *base, uint32_t refClockRate)

Returns the computed value for a frequency measurement cycle.

Parameters

- base – : SYSCON peripheral base address.
- refClockRate – : Reference clock rate used during the frequency measurement cycle.

Returns
Frequency in Hz.

FSL_FMEAS_DRIVER_VERSION

Defines LPC Frequency Measure driver version 2.1.1.

typedef FREQME_Type FMEAS_SYSCON_Type

FMEAS_SYSCON_FREQMECTRL_CAPVAL_MASK

FMEAS_SYSCON_FREQMECTRL_CAPVAL_SHIFT

FMEAS_SYSCON_FREQMECTRL_CAPVAL

FMEAS_SYSCON_FREQMECTRL_PROG_MASK

FMEAS_SYSCON_FREQMECTRL_PROG_SHIFT

FMEAS_SYSCON_FREQMECTRL_PROG

2.19 Hashcrypt: The Cryptographic Accelerator

2.20 Hashcrypt Background HASH

void HASHCRYPT_SHA_SetCallback(HASHCRYPT_Type *base, *hashcrypt_hash_ctx_t* *ctx, *hashcrypt_callback_t* callback, void *userData)

Initializes the HASHCRYPT handle for background hashing.

This function initializes the hash context for background hashing (Non-blocking) APIs. This is less typical interface to hash function, but can be used for parallel processing, when main CPU has something else to do. Example is digital signature RSASSA-PKCS1-V1_5-VERIFY((n,e),M,S) algorithm, where background hashing of M can be started, then CPU can compute S^e mod n (in parallel with background hashing) and once the digest becomes available, CPU can proceed to comparison of EM with EM'.

Parameters

- base – HASHCRYPT peripheral base address.
- ctx – **[out]** Hash context.

- callback – Callback function.
- userData – User data (to be passed as an argument to callback function, once callback is invoked from isr).

status_t HASHCRYPT_SHA_UpdateNonBlocking(HASHCRYPT_Type *base, *hashcrypt_hash_ctx_t* *ctx, const uint8_t *input, size_t inputSize)

Create running hash on given data.

Configures the HASHCRYPT to compute new running hash as AHB master and returns immediately. HASHCRYPT AHB Master mode supports only aligned input address and can be called only once per continuous block of data. Every call to this function must be preceded with *HASHCRYPT_SHA_Init()* and finished with *HASHCRYPT_SHA_Finish()*. Once callback function is invoked by HASHCRYPT isr, it should set a flag for the main application to finalize the hashing (padding) and to read out the final digest by calling *HASHCRYPT_SHA_Finish()*.

Parameters

- base – HASHCRYPT peripheral base address
- ctx – Specifies callback. Last incomplete 512-bit block of the input is copied into clear buffer for padding.
- input – 32-bit word aligned pointer to Input data.
- inputSize – Size of input data in bytes (must be word aligned)

Returns

Status of the hash update operation.

2.21 Hashcrypt common functions

FSL_HASHCRYPT_DRIVER_VERSION

HASHCRYPT driver version. Version 2.2.18.

Current version: 2.2.18

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Support loading AES key from unaligned address
- Version 2.0.2
  - Support loading AES key from unaligned address for different compiler and core variants
- Version 2.0.3
  - Remove SHA512 and AES ICB algorithm definitions
- Version 2.0.4
  - Add SHA context switch support
- Version 2.1.0
  - Update the register name and macro to align with new header.
- Version 2.1.1

  - Fix MISRA C-2012.
- Version 2.1.2
  - Support loading AES input data from unaligned address.
- Version 2.1.3
  - Fix MISRA C-2012.
- Version 2.1.4
  - Fix context switch cannot work when switching from AES.
- Version 2.1.5
  - Add data synchronization barrier inside hashcrypt_sha_ldm_stm_16_words() to prevent possible optimization issue.
- Version 2.2.0
  - Add AES-OFB and AES-CFB mixed IP/SW modes.
- Version 2.2.1
  - Add data synchronization barrier inside hashcrypt_sha_ldm_stm_16_words() prevent compiler from reordering memory write when -O2 or higher is used.
- Version 2.2.2
  - Add data synchronization barrier inside hashcrypt_sha_ldm_stm_16_words() to fix optimization issue
- Version 2.2.3
  - Added check for size in hashcrypt_aes_one_block to prevent overflowing COUNT field in MEMCTRL register, if its bigger than COUNT field do a multiple runs.
- Version 2.2.4
  - In all HASHCRYPT_AES_xx functions have been added setting CTRL_MODE bitfield to 0 after processing data, which decreases power consumption.
- Version 2.2.5
  - Add data synchronization barrier and instruction synchronization barrier inside hashcrypt_sha_process_message_data() to fix optimization issue
- Version 2.2.6
  - Add data synchronization barrier inside *HASHCRYPT_SHA_Update()* and hashcrypt_get_data() function to fix optimization issue on MDK and ARMGCC release targets
- Version 2.2.7
  - Add data synchronization barrier inside *HASHCRYPT_SHA_Update()* to fix optimization issue on MCUX IDE release target
- Version 2.2.8
  - Unify hashcrypt hashing behavior between aligned and unaligned input data
- Version 2.2.9
  - Add handling of set ERROR bit in the STATUS register
- Version 2.2.10
  - Fix missing error statement in hashcrypt_save_running_hash()
- Version 2.2.11
  - Fix incorrect SHA-256 calculation for long messages with reload

- Version 2.2.12
  - Fix hardfault issue on the Keil compiler due to unaligned memcpy() input on some optimization levels
- Version 2.2.13
  - Added function hashcrypt_seed_prng() which loading random number into PRNG_SEED register before AES operation for SCA protection
- Version 2.2.14
  - Modify function hashcrypt_get_data() to prevent issue with unaligned access
- Version 2.2.15
  - Add wait on DIGEST BIT inside hashcrypt_sha_one_block() to fix issues with some optimization flags
- Version 2.2.16
  - Add DSB instruction inside hashcrypt_sha_ldm_stm_16_words() to fix issues with some optimization flags
- Version 2.2.17
  - Fix context size when hashcrypt built with reload feature
- Version 2.2.18
  - Rename internal BUILD_ASSERT macro to HASHCRYPT_BUILD_ASSERT to avoid conflicts with environment-specific implementations.

enum _hashcrypt_algo_t

Algorithm used for Hashcrypt operation.

Values:

enumerator kHASHCRYPT_Sha1

SHA_1

enumerator kHASHCRYPT_Sha256

SHA_256

enumerator kHASHCRYPT_Aes

AES

typedef enum *_hashcrypt_algo_t* hashcrypt_algo_t

Algorithm used for Hashcrypt operation.

void HASHCRYPT_Init(HASHCRYPT_Type *base)

Enables clock and disables reset for HASHCRYPT peripheral.

Enable clock and disable reset for HASHCRYPT.

Parameters

- base – HASHCRYPT base address

void HASHCRYPT_Deinit(HASHCRYPT_Type *base)

Disables clock for HASHCRYPT peripheral.

Disable clock and enable reset.

Parameters

- base – HASHCRYPT base address

HASHCRYPT_MODE_SHA1
Algorithm definitions correspond with the values for Mode field in Control register !

HASHCRYPT_MODE_SHA256

HASHCRYPT_MODE_AES

2.22 Hashcrypt AES

enum _hashcrypt_aes_mode_t
AES mode.
Values:

enumerator kHASHCRYPT_AesEcb
AES ECB mode

enumerator kHASHCRYPT_AesCbc
AES CBC mode

enumerator kHASHCRYPT_AesCtr
AES CTR mode

enum _hashcrypt_aes_keysize_t
Size of AES key.
Values:

enumerator kHASHCRYPT_Aes128
AES 128 bit key

enumerator kHASHCRYPT_Aes192
AES 192 bit key

enumerator kHASHCRYPT_Aes256
AES 256 bit key

enumerator kHASHCRYPT_InvalidKey
AES invalid key

enum _hashcrypt_key
HASHCRYPT key source selection.
Values:

enumerator kHASHCRYPT_UserKey
HASHCRYPT user key

enumerator kHASHCRYPT_SecretKey

HASHCRYPT secret key (dedicated hw bus from PUF)

typedef enum *_hashcrypt_aes_mode_t* hashcrypt_aes_mode_t

AES mode.

typedef enum *_hashcrypt_aes_keysize_t* hashcrypt_aes_keysize_t

Size of AES key.

typedef enum *_hashcrypt_key* hashcrypt_key_t

HASHCRYPT key source selection.

typedef struct *_hashcrypt_handle* hashcrypt_handle_t

struct _hashcrypt_handle __attribute__ ((aligned))

status_t HASHCRYPT_AES_SetKey(HASHCRYPT_Type *base, *hashcrypt_handle_t* *handle, const uint8_t *key, size_t keySize)

Set AES key to hashcrypt_handle_t struct and optionally to HASHCRYPT.

Sets the AES key for encryption/decryption with the hashcrypt_handle_t structure. The hashcrypt_handle_t input argument specifies key source.

Parameters

- base – HASHCRYPT peripheral base address.
- handle – Handle used for the request.
- key – 0-mod-4 aligned pointer to AES key.
- keySize – AES key size in bytes. Shall equal 16, 24 or 32.

Returns

status from set key operation

status_t HASHCRYPT_AES_EncryptEcb(HASHCRYPT_Type *base, *hashcrypt_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size)

Encrypts AES on one or multiple 128-bit block(s).

Encrypts AES. The source plaintext and destination ciphertext can overlap in system memory.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.

Returns

Status from encrypt operation

status_t HASHCRYPT_AES_DecryptEcb(HASHCRYPT_Type *base, *hashcrypt_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size)

Decrypts AES on one or multiple 128-bit block(s).

Decrypts AES. The source ciphertext and destination plaintext can overlap in system memory.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- ciphertext – Input plain text to encrypt
- plaintext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.

Returns

Status from decrypt operation

status_t HASHCRYPT_AES_EncryptCbc(HASHCRYPT_Type *base, *hashcrypt_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[16])

Encrypts AES using CBC block mode.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- plaintext – Input plain text to encrypt
- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.

Returns

Status from encrypt operation

status_t HASHCRYPT_AES_DecryptCbc(HASHCRYPT_Type *base, *hashcrypt_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[16])

Decrypts AES using CBC block mode.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plain text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.

Returns

Status from decrypt operation

status_t HASHCRYPT_AES_CryptCtr(HASHCRYPT_Type *base, *hashcrypt_handle_t* *handle, const uint8_t *input, uint8_t *output, size_t size, uint8_t counter[16U], uint8_t counterlast[16U], size_t *szLeft)

Encrypts or decrypts AES using CTR block mode.

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- input – Input data for CTR block mode
- output – **[out]** Output data for CTR block mode
- size – Size of input and output data in bytes
- counter – **[inout]** Input counter (updates on return)
- counterlast – **[out]** Output cipher of last counter, for chained CTR calls (statefull encryption). NULL can be passed if chained calls are not used.
- szLeft – **[out]** Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns

Status from encrypt operation

status_t HASHCRYPT_AES_CryptOfb(HASHCRYPT_Type *base, *hashcrypt_handle_t* *handle, const uint8_t *input, uint8_t *output, size_t size, const uint8_t iv[16U])

Encrypts or decrypts AES using OFB block mode.

Encrypts or decrypts AES using OFB block mode. AES OFB mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- input – Input data for OFB block mode
- output – **[out]** Output data for OFB block mode
- size – Size of input and output data in bytes
- iv – Input initial vector to combine with the first input block.

Returns

Status from encrypt operation

status_t HASHCRYPT_AES_EncryptCfb(HASHCRYPT_Type *base, *hashcrypt_handle_t* *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[16])

Encrypts AES using CFB block mode.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- plaintext – Input plain text to encrypt

- ciphertext – **[out]** Output cipher text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.

Returns
Status from encrypt operation

status_t HASHCRYPT_AES_DecryptCfb(HASHCRYPT_Type *base, *hashcrypt_handle_t* *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[16])

Decrypts AES using CFB block mode.

Parameters

- base – HASHCRYPT peripheral base address
- handle – Handle used for this request.
- ciphertext – Input cipher text to decrypt
- plaintext – **[out]** Output plaintext text
- size – Size of input and output data in bytes. Must be multiple of 16 bytes.
- iv – Input initial vector to combine with the first input block.

Returns
Status from encrypt operation

HASHCRYPT_AES_BLOCK_SIZE

AES block size in bytes

AES_ENCRYPT

AES_DECRYPT

struct _hashcrypt_handle

#include <fsl_hashcrypt.h> Specify HASHCRYPT's key resource.

Public Members

uint32_t keyWord[8]

Copy of user key (set by *HASHCRYPT_AES_SetKey()*.

hashcrypt_key_t keyType

For operations with key (such as AES encryption/decryption), specify key type.

2.23 Hashcrypt HASH

typedef struct *_hashcrypt_hash_ctx_t* hashcrypt_hash_ctx_t

Storage type used to save hash context.

typedef void (*hashcrypt_callback_t)(HASHCRYPT_Type *base, *hashcrypt_hash_ctx_t* *ctx, *status_t* status, void *userData)

HASHCRYPT background hash callback function.

status_t HASHCRYPT_SHA(HASHCRYPT_Type *base, *hashcrypt_algo_t* algo, const uint8_t *input, size_t inputSize, uint8_t *output, size_t *outputSize)

Create HASH on given data.

Perform the full SHA in one function call. The function is blocking.

Parameters

- base – HASHCRYPT peripheral base address
- algo – Underlaying algorithm to use for hash computation.
- input – Input data
- inputSize – Size of input data in bytes
- output – **[out]** Output hash data
- outputSize – **[out]** Output parameter storing the size of the output hash in bytes

Returns

Status of the one call hash operation.

status_t HASHCRYPT_SHA_Init(HASHCRYPT_Type *base, *hashcrypt_hash_ctx_t* *ctx, *hashcrypt_algo_t* algo)

Initialize HASH context.

This function initializes the HASH.

Parameters

- base – HASHCRYPT peripheral base address
- ctx – **[out]** Output hash context
- algo – Underlaying algorithm to use for hash computation.

Returns

Status of initialization

status_t HASHCRYPT_SHA_Update(HASHCRYPT_Type *base, *hashcrypt_hash_ctx_t* *ctx, const uint8_t *input, size_t inputSize)

Add data to current HASH.

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed. The functions blocks. If it returns kStatus_Success, the running hash has been updated (HASHCRYPT has processed the input data), so the memory at input pointer can be released back to system. The HASHCRYPT context buffer is updated with the running hash and with all necessary information to support possible context switch.

Parameters

- base – HASHCRYPT peripheral base address
- ctx – **[inout]** HASH context
- input – Input data
- inputSize – Size of input data in bytes

Returns

Status of the hash update operation

status_t HASHCRYPT_SHA_Finish(HASHCRYPT_Type *base, *hashcrypt_hash_ctx_t* *ctx, uint8_t *output, size_t *outputSize)

Finalize hashing.

Outputs the final hash (computed by HASHCRYPT_HASH_Update()) and erases the context.

Parameters

- base – HASHCRYPT peripheral base address
- ctx – **[inout]** Input hash context
- output – **[out]** Output hash data
- outputSize – **[inout]** Optional parameter (can be passed as NULL). On function entry, it specifies the size of output[] buffer. On function return, it stores the number of updated output bytes.

Returns

Status of the hash finish operation

HASHCRYPT_HASH_CTX_SIZE

HASHCRYPT HASH Context size.

struct __hashcrypt_hash_ctx_t

#include <fsl_hashcrypt.h> Storage type used to save hash context.

Public Members

uint32_t x[31]

storage

2.24 I2C: Inter-Integrated Circuit Driver

2.25 I2C DMA Driver

void I2C_MasterTransferCreateHandleDMA(I2C_Type *base, *i2c_master_dma_handle_t* *handle, *i2c_master_dma_transfer_callback_t* callback, void *userData, *dma_handle_t* *dmaHandle)

Init the I2C handle which is used in transactional functions.

Parameters

- base – I2C peripheral base address
- handle – pointer to i2c_master_dma_handle_t structure
- callback – pointer to user callback function
- userData – user param passed to the callback function
- dmaHandle – DMA handle pointer

status_t I2C_MasterTransferDMA(I2C_Type *base, *i2c_master_dma_handle_t* *handle, *i2c_master_transfer_t* *xfer)

Performs a master dma non-blocking transfer on the I2C bus.

Parameters

- base – I2C peripheral base address
- handle – pointer to i2c_master_dma_handle_t structure
- xfer – pointer to transfer structure of i2c_master_transfer_t

Return values

- kStatus_Success – Sucessully complete the data transmission.
- kStatus_I2C_Busy – Previous transmission still not finished.
- kStatus_I2C_Timeout – Transfer error, wait signal timeout.
- kStatus_I2C_ArbitrationLost – Transfer error, arbitration lost.
- kStataus_I2C_Nak – Transfer error, receive Nak during transfer.

status_t I2C_MasterTransferGetCountDMA(I2C_Type *base, *i2c_master_dma_handle_t* *handle, size_t *count)

Get master transfer status during a dma non-blocking transfer.

Parameters

- base – I2C peripheral base address
- handle – pointer to i2c_master_dma_handle_t structure
- count – Number of bytes transferred so far by the non-blocking transaction.

void I2C_MasterTransferAbortDMA(I2C_Type *base, *i2c_master_dma_handle_t* *handle)

Abort a master dma non-blocking transfer in a early time.

Parameters

- base – I2C peripheral base address
- handle – pointer to i2c_master_dma_handle_t structure

FSL_I2C_DMA_DRIVER_VERSION

I2C DMA driver version.

typedef struct *_i2c_master_dma_handle* i2c_master_dma_handle_t

I2C master dma handle typedef.

typedef void (*i2c_master_dma_transfer_callback_t)(I2C_Type *base, *i2c_master_dma_handle_t* *handle, *status_t* status, void *userData)

I2C master dma transfer callback typedef.

typedef void (*flexcomm_i2c_dma_master_irq_handler_t)(I2C_Type *base, *i2c_master_dma_handle_t* *handle)

Typedef for master dma handler.

I2C_MAX_DMA_TRANSFER_COUNT

Maximum lenght of single DMA transfer (determined by capability of the DMA engine)

struct _i2c_master_dma_handle

#include <fsl_i2c_dma.h> I2C master dma transfer structure.

Public Members

uint8_t state
Transfer state machine current state.

uint32_t transferCount
Indicates progress of the transfer

uint32_t remainingBytesDMA
Remaining byte count to be transferred using DMA.

uint8_t *buf
Buffer pointer for current state.

bool checkAddrNack
Whether to check the nack signal is detected during addressing.

dma_handle_t *dmaHandle
The DMA handler used.

i2c_master_transfer_t transfer
Copy of the current transfer info.

i2c_master_dma_transfer_callback_t completionCallback
Callback function called after dma transfer finished.

void *userData
Callback parameter passed to callback function.

2.26 I2C Driver

FSL_I2C_DRIVER_VERSION
I2C driver version.

I2C status return codes.

Values:

enumerator kStatus_I2C_Busy
The master is already performing a transfer.

enumerator kStatus_I2C_Idle
The slave driver is idle.

enumerator kStatus_I2C_Nak
The slave device sent a NAK in response to a byte.

enumerator kStatus_I2C_InvalidParameter
Unable to proceed due to invalid parameter.

enumerator kStatus_I2C_BitError
Transferred bit was not seen on the bus.

enumerator kStatus_I2C_ArbitrationLost
Arbitration lost error.

enumerator kStatus_I2C_NoTransferInProgress
Attempt to abort a transfer when one is not in progress.

enumerator kStatus_I2C_DmaRequestFail
DMA request failed.

enumerator kStatus_I2C_StartStopError
Start and stop error.

enumerator kStatus_I2C_UnexpectedState
Unexpected state.

enumerator kStatus_I2C_Timeout
Timeout when waiting for I2C master/slave pending status to set to continue transfer.

enumerator kStatus_I2C_Addr_Nak
NAK received for Address

enumerator kStatus_I2C_EventTimeout
Timeout waiting for bus event.

enumerator kStatus_I2C_SclLowTimeout
Timeout SCL signal remains low.

enum _i2c_status_flags
I2C status flags.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI2C_MasterPendingFlag
The I2C module is waiting for software interaction. bit 0

enumerator kI2C_MasterArbitrationLostFlag
The arbitration of the bus was lost. There was collision on the bus. bit 4

enumerator kI2C_MasterStartStopErrorFlag
There was an error during start or stop phase of the transaction. bit 6

enumerator kI2C_MasterIdleFlag
The I2C master idle status. bit 5

enumerator kI2C_MasterRxReadyFlag
The I2C master rx ready status. bit 1

enumerator kI2C_MasterTxReadyFlag
The I2C master tx ready status. bit 2

enumerator kI2C_MasterAddrNackFlag
The I2C master address nack status. bit 7

enumerator kI2C_MasterDataNackFlag
The I2C master data nack status. bit 3

enumerator kI2C_SlavePendingFlag
The I2C module is waiting for software interaction. bit 8

enumerator kI2C_SlaveNotStretching
Indicates whether the slave is currently stretching clock (0 = yes, 1 = no). bit 11

enumerator kI2C_SlaveSelected
Indicates whether the slave is selected by an address match. bit 14

enumerator kI2C_SaveDeselected
Indicates that slave was previously deselected (deselect event took place, w1c). bit 15

enumerator kI2C_SlaveAddressedFlag
One of the I2C slave's 4 addresses is matched. bit 22

enumerator kI2C_SlaveReceiveFlag
Slave receive data available. bit 9

enumerator kI2C_SlaveTransmitFlag
Slave data can be transmitted. bit 10

enumerator kI2C_SlaveAddress0MatchFlag
Slave address0 match. bit 20

enumerator kI2C_SlaveAddress1MatchFlag
Slave address1 match. bit 12

enumerator kI2C_SlaveAddress2MatchFlag
Slave address2 match. bit 13

enumerator kI2C_SlaveAddress3MatchFlag
Slave address3 match. bit 21

enumerator kI2C_MonitorReadyFlag
The I2C monitor ready interrupt. bit 16

enumerator kI2C_MonitorOverflowFlag
The monitor data overrun interrupt. bit 17

enumerator kI2C_MonitorActiveFlag
The monitor is active. bit 18

enumerator kI2C_MonitorIdleFlag
The monitor idle interrupt. bit 19

enumerator kI2C_EventTimeoutFlag
The bus event timeout interrupt. bit 24

enumerator kI2C_SclTimeoutFlag
The SCL timeout interrupt. bit 25

enumerator kI2C_MasterAllClearFlags

enumerator kI2C_SlaveAllClearFlags

enumerator kI2C_CommonAllClearFlags

enum _i2c_interrupt_enable
I2C interrupt enable.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI2C_MasterPendingInterruptEnable
The I2C master communication pending interrupt.

enumerator kI2C_MasterArbitrationLostInterruptEnable
The I2C master arbitration lost interrupt.

enumerator kI2C_MasterStartStopErrorInterruptEnable
The I2C master start/stop timing error interrupt.

enumerator kI2C_SlavePendingInterruptEnable
The I2C slave communication pending interrupt.

enumerator kI2C_SlaveNotStretchingInterruptEnable
The I2C slave not streching interrupt, deep-sleep mode can be entered only when this interrupt occurs.

enumerator kI2C_SlaveDeselectedInterruptEnable
The I2C slave deselection interrupt.

enumerator kI2C_MonitorReadyInterruptEnable
The I2C monitor ready interrupt.

enumerator kI2C_MonitorOverflowInterruptEnable
The monitor data overrun interrupt.

enumerator kI2C_MonitorIdleInterruptEnable
The monitor idle interrupt.

enumerator kI2C_EventTimeoutInterruptEnable
The bus event timeout interrupt.

enumerator kI2C_SclTimeoutInterruptEnable
The SCL timeout interrupt.

enumerator kI2C_MasterAllInterruptEnable

enumerator kI2C_SlaveAllInterruptEnable

enumerator kI2C_CommonAllInterruptEnable

I2C_RETRY_TIMES
Retry times for waiting flag.

I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK
Whether to ignore the nack signal of the last byte during master transmit.

I2C_STAT_MSTCODE_IDLE
Master Idle State Code

I2C_STAT_MSTCODE_RXREADY
Master Receive Ready State Code

I2C_STAT_MSTCODE_TXREADY
Master Transmit Ready State Code

I2C_STAT_MSTCODE_NACKADR
Master NACK by slave on address State Code

I2C_STAT_MSTCODE_NACKDAT
Master NACK by slave on data State Code

I2C_STAT_SLVST_ADDR

I2C_STAT_SLVST_RX

I2C_STAT_SLVST_TX

2.27 I2C Master Driver

void I2C_MasterGetDefaultConfig(*i2c_master_config_t* *masterConfig)

Provides a default configuration for the I2C master peripheral.

This function provides the following default configuration for the I2C master peripheral:

```
masterConfig->enableMaster            = true;
masterConfig->baudRate_Bps            = 100000U;
masterConfig->enableTimeout           = false;
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with *I2C_MasterInit()*.

Parameters

- masterConfig – **[out]** User provided configuration structure for default values. Refer to *i2c_master_config_t*.

void I2C_MasterInit(I2C_Type *base, const *i2c_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Initializes the I2C master peripheral.

This function enables the peripheral clock and initializes the I2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

- base – The I2C peripheral base address.
- masterConfig – User provided peripheral configuration. Use *I2C_MasterGetDefaultConfig()* to get a set of defaults that you can override.
- srcClock_Hz – Frequency in Hertz of the I2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

void I2C_MasterDeinit(I2C_Type *base)

Deinitializes the I2C master peripheral.

This function disables the I2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The I2C peripheral base address.

uint32_t I2C_GetInstance(I2C_Type *base)

Returns an instance number given a base address.

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

- base – The I2C peripheral base address.

Returns

I2C instance number starting from 0.

static inline void I2C_MasterReset(I2C_Type *base)

Performs a software reset.

Restores the I2C master peripheral to reset conditions.

Parameters

- base – The I2C peripheral base address.

static inline void I2C_MasterEnable(I2C_Type *base, bool enable)

Enables or disables the I2C module as master.

Parameters

- base – The I2C peripheral base address.
- enable – Pass true to enable or false to disable the specified I2C as master.

uint32_t I2C_GetStatusFlags(I2C_Type *base)

Gets the I2C status flags.

A bit mask with the state of all I2C status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

See also:

_i2c_status_flags.

Parameters

- base – The I2C 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 I2C_ClearStatusFlags(I2C_Type *base, uint32_t statusMask)

Clears the I2C status flag state.

Refer to kI2C_CommonAllClearStatusFlags, kI2C_MasterAllClearStatusFlags and kI2C_SlaveAllClearStatusFlags to see the clearable flags. Attempts to clear other flags has no effect.

See also:

_i2c_status_flags, _i2c_master_status_flags and _i2c_slave_status_flags.

Parameters

- base – The I2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of the members in kI2C_CommonAllClearStatusFlags, kI2C_MasterAllClearStatusFlags and kI2C_SlaveAllClearStatusFlags. You may pass the result of a previous call to *I2C_GetStatusFlags()*.

static inline void I2C_MasterClearStatusFlags(I2C_Type *base, uint32_t statusMask)

Clears the I2C master status flag state.

Deprecated:

Do not use this function. It has been superceded by *I2C_ClearStatusFlags* The following status register flags can be cleared:

- *kI2C_MasterArbitrationLostFlag*
- *kI2C_MasterStartStopErrorFlag*

Attempts to clear other flags has no effect.

See also:

_i2c_status_flags.

Parameters

- base – The I2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of *_i2c_status_flags* enumerators OR'd together. You may pass the result of a previous call to *I2C_GetStatusFlags()*.

static inline void I2C_EnableInterrupts(I2C_Type *base, uint32_t interruptMask)

Enables the I2C interrupt requests.

Parameters

- base – The I2C peripheral base address.
- interruptMask – Bit mask of interrupts to enable. See *_i2c_interrupt_enable* for the set of constants that should be OR'd together to form the bit mask.

static inline void I2C_DisableInterrupts(I2C_Type *base, uint32_t interruptMask)

Disables the I2C interrupt requests.

Parameters

- base – The I2C peripheral base address.
- interruptMask – Bit mask of interrupts to disable. See *_i2c_interrupt_enable* for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I2C_GetEnabledInterrupts(I2C_Type *base)

Returns the set of currently enabled I2C interrupt requests.

Parameters

- base – The I2C peripheral base address.

Returns

A bitmask composed of *_i2c_interrupt_enable* enumerators OR'd together to indicate the set of enabled interrupts.

void I2C_MasterSetBaudRate(I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the I2C bus frequency for master transactions.

The I2C 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.

Parameters

- base – The I2C peripheral base address.
- srcClock_Hz – I2C functional clock frequency in Hertz.
- baudRate_Bps – Requested bus frequency in bits per second.

void I2C_MasterSetTimeoutValue(I2C_Type *base, uint8_t timeout_Ms, uint32_t srcClock_Hz)

Sets the I2C bus timeout value.

If the SCL signal remains low or bus does not have event longer than the timeout value, kI2C_SclTimeoutFlag or kI2C_EventTimeoutFlag is set. This can indicete the bus is held by slave or any fault occurs to the I2C module.

Parameters

- base – The I2C peripheral base address.
- timeout_Ms – Timeout value in millisecond.
- srcClock_Hz – I2C functional clock frequency in Hertz.

static inline bool I2C_MasterGetBusIdleState(I2C_Type *base)

Returns whether the bus is idle.

Requires the master mode to be enabled.

Parameters

- base – The I2C peripheral base address.

Return values

- true – Bus is busy.
- false – Bus is idle.

status_t I2C_MasterStart(I2C_Type *base, uint8_t address, *i2c_direction_t* direction)

Sends a START on the I2C bus.

This function is used to initiate a new master mode transfer by sending the START signal. The slave address is sent following the I2C START signal.

Parameters

- base – I2C peripheral base pointer
- address – 7-bit slave device address.
- direction – Master transfer directions(transmit/receive).

Return values

- kStatus_Success – Successfully send the start signal.
- kStatus_I2C_Busy – Current bus is busy.

status_t I2C_MasterStop(I2C_Type *base)

Sends a STOP signal on the I2C bus.

Return values

- kStatus_Success – Successfully send the stop signal.
- kStatus_I2C_Timeout – Send stop signal failed, timeout.

static inline *status_t* I2C_MasterRepeatedStart(I2C_Type *base, uint8_t address, *i2c_direction_t* direction)

Sends a REPEATED START on the I2C bus.

Parameters

- base – I2C peripheral base pointer
- address – 7-bit slave device address.
- direction – Master transfer directions(transmit/receive).

Return values

- kStatus_Success – Successfully send the start signal.
- kStatus_I2C_Busy – Current bus is busy but not occupied by current I2C master.

status_t I2C_MasterWriteBlocking(I2C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)

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_I2C_Nak*.

Parameters

- base – The I2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.
- flags – Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag

Return values

- kStatus_Success – Data was sent successfully.
- kStatus_I2C_Busy – Another master is currently utilizing the bus.
- kStatus_I2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_I2C_ArbitrationLost – Arbitration lost error.

status_t I2C_MasterReadBlocking(I2C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)

Performs a polling receive transfer on the I2C bus.

Parameters

- base – The I2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.
- flags – Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag

Return values

- kStatus_Success – Data was received successfully.
- kStatus_I2C_Busy – Another master is currently utilizing the bus.
- kStatus_I2C_Nak – The slave device sent a NAK in response to a byte.
- kStatus_I2C_ArbitrationLost – Arbitration lost error.

status_t I2C_MasterTransferBlocking(I2C_Type *base, *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 arbitration lost or receiving a NAK.

Parameters

- base – I2C peripheral base address.
- xfer – Pointer to the transfer structure.

Return values

- kStatus_Success – Successfully complete the data transmission.
- kStatus_I2C_Busy – Previous transmission still not finished.
- kStatus_I2C_Timeout – Transfer error, wait signal timeout.
- kStatus_I2C_ArbitrationLost – Transfer error, arbitration lost.
- kStataus_I2C_Nak – Transfer error, receive NAK during transfer.
- kStataus_I2C_Addr_Nak – Transfer error, receive NAK during addressing.

void I2C_MasterTransferCreateHandle(I2C_Type *base, *i2c_master_handle_t* *handle, *i2c_master_transfer_callback_t* callback, void *userData)

Creates a new handle for the I2C 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 *I2C_MasterTransferAbort()* API shall be called.

Parameters

- base – The I2C peripheral base address.
- handle – **[out]** Pointer to the I2C master driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t I2C_MasterTransferNonBlocking(I2C_Type *base, *i2c_master_handle_t* *handle, *i2c_master_transfer_t* *xfer)

Performs a non-blocking transaction on the I2C bus.

Parameters

- base – The I2C peripheral base address.
- handle – Pointer to the I2C master driver handle.
- xfer – The pointer to the transfer descriptor.

Return values

- kStatus_Success – The transaction was started successfully.
- kStatus_I2C_Busy – Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

status_t I2C_MasterTransferGetCount(I2C_Type *base, *i2c_master_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – The I2C peripheral base address.
- handle – Pointer to the I2C master driver handle.
- count – **[out]** Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_I2C_Busy –

status_t I2C_MasterTransferAbort(I2C_Type *base, *i2c_master_handle_t* *handle)

Terminates a non-blocking I2C master transmission early.

Note: It is not safe to call this function from an IRQ handler that has a higher priority than the I2C peripheral's IRQ priority.

Parameters

- base – The I2C peripheral base address.
- handle – Pointer to the I2C master driver handle.

Return values

- kStatus_Success – A transaction was successfully aborted.
- kStatus_I2C_Timeout – Timeout during polling for flags.

void I2C_MasterTransferHandleIRQ(I2C_Type *base, *i2c_master_handle_t* *handle)

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 I2C peripheral base address.
- handle – Pointer to the I2C master driver handle.

enum _i2c_direction

Direction of master and slave transfers.

Values:

enumerator kI2C_Write

Master transmit.

enumerator kI2C_Read

Master receive.

enum _i2c_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 *_i2c_master_transfer::flags* field.

Values:

enumerator kI2C_TransferDefaultFlag

Transfer starts with a start signal, stops with a stop signal.

enumerator kI2C_TransferNoStartFlag

Don't send a start condition, address, and sub address

enumerator kI2C_TransferRepeatedStartFlag

Send a repeated start condition

enumerator kI2C_TransferNoStopFlag

Don't send a stop condition.

enum _i2c_transfer_states

States for the state machine used by transactional APIs.

Values:

enumerator kIdleState

enumerator kTransmitSubaddrState

enumerator kTransmitDataState

enumerator kReceiveDataBeginState

enumerator kReceiveDataState

enumerator kReceiveLastDataState

enumerator kStartState

enumerator kStopState

enumerator kWaitForCompletionState

typedef enum *_i2c_direction* i2c_direction_t

Direction of master and slave transfers.

typedef struct *_i2c_master_config* i2c_master_config_t

Structure with settings to initialize the I2C master module.

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the *I2C_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 *_i2c_master_transfer* i2c_master_transfer_t

I2C master transfer typedef.

typedef struct *_i2c_master_handle* i2c_master_handle_t

I2C master handle typedef.

typedef void (*i2c_master_transfer_callback_t)(I2C_Type *base, *i2c_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 *I2C_MasterTransferCreateHandle()*.

Param base
The I2C 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.

struct __i2c_master_config

#include <fsl_i2c.h> Structure with settings to initialize the I2C master module.

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the *I2C_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.

uint32_t baudRate_Bps

Desired baud rate in bits per second.

bool enableTimeout

Enable internal timeout function.

uint8_t timeout_Ms

Event timeout and SCL low timeout value.

struct __i2c_master_transfer

#include <fsl_i2c.h> Non-blocking transfer descriptor structure.

This structure is used to pass transaction parameters to the *I2C_MasterTransferNonBlocking()* API.

Public Members

uint32_t flags

Bit mask of options for the transfer. See enumeration *_i2c_master_transfer_flags* for available options. Set to 0 or *kI2C_TransferDefaultFlag* for normal transfers.

uint8_t slaveAddress

The 7-bit slave address.

i2c_direction_t direction
: Either *kI2C_Read* or *kI2C_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 __i2c_master_handle
: *#include <fsl_i2c.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 transferCount
: Indicates progress of the transfer

uint32_t remainingBytes
: Remaining byte count in current state.

uint8_t *buf
: Buffer pointer for current state.

bool checkAddrNack
: Whether to check the nack signal is detected during addressing.

i2c_master_transfer_t transfer
: Copy of the current transfer info.

i2c_master_transfer_callback_t completionCallback
: Callback function pointer.

void *userData
: Application data passed to callback.

2.28 I2C Slave Driver

void I2C_SlaveGetDefaultConfig(*i2c_slave_config_t* *slaveConfig)

Provides a default configuration for the I2C slave peripheral.

This function provides the following default configuration for the I2C slave peripheral:

```
slaveConfig->enableSlave = true;
slaveConfig->address0.disable = false;
slaveConfig->address0.address = 0u;
slaveConfig->address1.disable = true;
slaveConfig->address2.disable = true;
slaveConfig->address3.disable = true;
slaveConfig->busSpeed = kI2C_SlaveStandardMode;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with *I2C_SlaveInit()*. Be sure to override at least the *address0.address* 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 *i2c_slave_config_t*.

status_t I2C_SlaveInit(I2C_Type *base, const *i2c_slave_config_t* *slaveConfig, uint32_t srcClock_Hz)

Initializes the I2C slave peripheral.

This function enables the peripheral clock and initializes the I2C slave peripheral as described by the user provided configuration.

Parameters

- base – The I2C peripheral base address.
- slaveConfig – User provided peripheral configuration. Use *I2C_SlaveGetDefaultConfig()* to get a set of defaults that you can override.
- srcClock_Hz – Frequency in Hertz of the I2C functional clock. Used to calculate CLKDIV value to provide enough data setup time for master when slave stretches the clock.

void I2C_SlaveSetAddress(I2C_Type *base, *i2c_slave_address_register_t* addressRegister, uint8_t address, bool addressDisable)

Configures Slave Address n register.

This function writes new value to Slave Address register.

Parameters

- base – The I2C peripheral base address.
- addressRegister – The module supports multiple address registers. The parameter determines which one shall be changed.
- address – The slave address to be stored to the address register for matching.
- addressDisable – Disable matching of the specified address register.

void I2C_SlaveDeinit(I2C_Type *base)

Deinitializes the I2C slave peripheral.

This function disables the I2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

- base – The I2C peripheral base address.

static inline void I2C_SlaveEnable(I2C_Type *base, bool enable)

Enables or disables the I2C module as slave.

Parameters

- base – The I2C peripheral base address.
- enable – True to enable or flase to disable.

static inline void I2C_SlaveClearStatusFlags(I2C_Type *base, uint32_t statusMask)

Clears the I2C status flag state.

The following status register flags can be cleared:

- slave deselected flag

Attempts to clear other flags has no effect.

See also:

_i2c_slave_flags.

Parameters

- base – The I2C peripheral base address.
- statusMask – A bitmask of status flags that are to be cleared. The mask is composed of _i2c_slave_flags enumerators OR'd together. You may pass the result of a previous call to I2C_SlaveGetStatusFlags().

status_t I2C_SlaveWriteBlocking(I2C_Type *base, const uint8_t *txBuff, size_t txSize)

Performs a polling send transfer on the I2C bus.

The function executes blocking address phase and blocking data phase.

Parameters

- base – The I2C peripheral base address.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Returns

kStatus_Success Data has been sent.

Returns

kStatus_Fail Unexpected slave state (master data write while master read from slave is expected).

status_t I2C_SlaveReadBlocking(I2C_Type *base, uint8_t *rxBuff, size_t rxSize)

Performs a polling receive transfer on the I2C bus.

The function executes blocking address phase and blocking data phase.

Parameters

- base – The I2C peripheral base address.
- rxBuff – The pointer to the data to be transferred.
- rxSize – The length in bytes of the data to be transferred.

Returns

kStatus_Success Data has been received.

Returns

kStatus_Fail Unexpected slave state (master data read while master write to slave is expected).

void I2C_SlaveTransferCreateHandle(I2C_Type *base, *i2c_slave_handle_t* *handle, *i2c_slave_transfer_callback_t* callback, void *userData)

Creates a new handle for the I2C 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 *I2C_SlaveTransferAbort()* API shall be called.

Parameters

- base – The I2C peripheral base address.
- handle – **[out]** Pointer to the I2C slave driver handle.
- callback – User provided pointer to the asynchronous callback function.
- userData – User provided pointer to the application callback data.

status_t I2C_SlaveTransferNonBlocking(I2C_Type *base, *i2c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling *I2C_SlaveInit()* and *I2C_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 *I2C_SlaveTransferCreateHandle()*. The callback is always invoked from the interrupt context.

If no slave Tx transfer is busy, a master read from slave request invokes *kI2C_SlaveTransmitEvent* callback. If no slave Rx transfer is busy, a master write to slave request invokes *kI2C_SlaveReceiveEvent* callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *i2c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kI2C_SlaveTransmitEvent* and *kI2C_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 *kI2C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- base – The I2C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure which stores the transfer state.
- eventMask – Bit mask formed by OR'ing together *i2c_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 *kI2C_SlaveAllEvents* to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_I2C_Busy – Slave transfers have already been started on this handle.

status_t I2C_SlaveSetSendBuffer(I2C_Type *base, volatile *i2c_slave_transfer_t* *transfer, const void *txData, size_t txSize, uint32_t eventMask)

Starts accepting master read from slave requests.

The function can be called in response to *kI2C_SlaveTransmitEvent* callback to start a new slave Tx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *i2c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kI2C_SlaveTransmitEvent* and *kI2C_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 *kI2C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- base – The I2C peripheral base address.
- transfer – Pointer to *i2c_slave_transfer_t* structure.
- txData – Pointer to data to send to master.
- txSize – Size of txData in bytes.
- eventMask – Bit mask formed by OR'ing together *i2c_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 *kI2C_SlaveAllEvents* to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_I2C_Busy – Slave transfers have already been started on this handle.

status_t I2C_SlaveSetReceiveBuffer(I2C_Type *base, volatile *i2c_slave_transfer_t* *transfer, void *rxData, size_t rxSize, uint32_t eventMask)

Starts accepting master write to slave requests.

The function can be called in response to *kI2C_SlaveReceiveEvent* callback to start a new slave Rx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *i2c_slave_transfer_event_t* enumerators for the events you wish to receive. The *kI2C_SlaveTransmitEvent* and *kI2C_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 *kI2C_SlaveAllEvents* constant is provided as a convenient way to enable all events.

Parameters

- base – The I2C peripheral base address.
- transfer – Pointer to *i2c_slave_transfer_t* structure.
- rxData – Pointer to data to store data from master.
- rxSize – Size of rxData in bytes.
- eventMask – Bit mask formed by OR'ing together *i2c_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 *kI2C_SlaveAllEvents* to enable all events.

Return values

- kStatus_Success – Slave transfers were successfully started.
- kStatus_I2C_Busy – Slave transfers have already been started on this handle.

static inline uint32_t I2C_SlaveGetReceivedAddress(I2C_Type *base, volatile *i2c_slave_transfer_t* *transfer)

Returns the slave address sent by the I2C master.

This function should only be called from the address match event callback *kI2C_SlaveAddressMatchEvent*.

Parameters

- base – The I2C peripheral base address.
- transfer – The I2C slave transfer.

Returns

The 8-bit address matched by the I2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

void I2C_SlaveTransferAbort(I2C_Type *base, *i2c_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 I2C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure which stores the transfer state.

Return values

- kStatus_Success –
- kStatus_I2C_Idle –

status_t I2C_SlaveTransferGetCount(I2C_Type *base, *i2c_slave_handle_t* *handle, size_t *count)

Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_slave_handle_t structure.
- 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 I2C_SlaveTransferHandleIRQ(I2C_Type *base, *i2c_slave_handle_t* *handle)

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 I2C peripheral base address.
- handle – Pointer to i2c_slave_handle_t structure which stores the transfer state.

enum _i2c_slave_address_register

I2C slave address register.

Values:

enumerator kI2C_SlaveAddressRegister0

Slave Address 0 register.

enumerator kI2C_SlaveAddressRegister1

Slave Address 1 register.

enumerator kI2C_SlaveAddressRegister2

Slave Address 2 register.

enumerator kI2C_SlaveAddressRegister3

Slave Address 3 register.

enum _i2c_slave_address_qual_mode

I2C slave address match options.

Values:

enumerator kI2C_QualModeMask

The SLVQUAL0 field (qualAddress) is used as a logical mask for matching address0.

enumerator kI2C_QualModeExtend

The SLVQUAL0 (qualAddress) field is used to extend address 0 matching in a range of addresses.

enum _i2c_slave_bus_speed

I2C slave bus speed options.

Values:

enumerator kI2C_SlaveStandardMode

enumerator kI2C_SlaveFastMode

enumerator kI2C_SlaveFastModePlus

enumerator kI2C_SlaveHsMode

enum _i2c_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 *I2C_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 kI2C_SlaveAddressMatchEvent
Received the slave address after a start or repeated start.

enumerator kI2C_SlaveTransmitEvent
Callback is requested to provide data to transmit (slave-transmitter role).

enumerator kI2C_SlaveReceiveEvent
Callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI2C_SlaveCompletionEvent
All data in the active transfer have been consumed.

enumerator kI2C_SlaveDeselectedEvent
The slave function has become deselected (SLVSEL flag changing from 1 to 0.

enumerator kI2C_SlaveAllEvents
Bit mask of all available events.

enum _i2c_slave_fsm
I2C slave software finite state machine states.

Values:

enumerator kI2C_SlaveFsmAddressMatch

enumerator kI2C_SlaveFsmReceive

enumerator kI2C_SlaveFsmTransmit

typedef enum *_i2c_slave_address_register* i2c_slave_address_register_t
I2C slave address register.

typedef struct *_i2c_slave_address* i2c_slave_address_t
Data structure with 7-bit Slave address and Slave address disable.

typedef enum *_i2c_slave_address_qual_mode* i2c_slave_address_qual_mode_t
I2C slave address match options.

typedef enum *_i2c_slave_bus_speed* i2c_slave_bus_speed_t
I2C slave bus speed options.

typedef struct *_i2c_slave_config* i2c_slave_config_t
Structure with settings to initialize the I2C slave module.

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the *I2C_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 *_i2c_slave_transfer_event* i2c_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 *I2C_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 *_i2c_slave_handle* i2c_slave_handle_t

I2C slave handle typedef.

typedef struct *_i2c_slave_transfer* i2c_slave_transfer_t

I2C slave transfer structure.

typedef void (*i2c_slave_transfer_callback_t)(I2C_Type *base, volatile *i2c_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 I2C_SlaveSetCallback() function after you have created a handle.

Param base
Base address for the I2C 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 enum *_i2c_slave_fsm* i2c_slave_fsm_t

I2C slave software finite state machine states.

typedef void (*flexcomm_i2c_master_irq_handler_t)(I2C_Type *base, *i2c_master_handle_t* *handle)

Typedef for master interrupt handler.

typedef void (*flexcomm_i2c_slave_irq_handler_t)(I2C_Type *base, *i2c_slave_handle_t* *handle)

Typedef for slave interrupt handler.

struct i2c_slave_address

#include <fsl_i2c.h> Data structure with 7-bit Slave address and Slave address disable.

Public Members

uint8_t address

7-bit Slave address SLVADR.

bool addressDisable

Slave address disable SADISABLE.

struct _i2c_slave_config

#include <fsl_i2c.h> Structure with settings to initialize the I2C slave module.

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the *I2C_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

i2c_slave_address_t address0

Slave's 7-bit address and disable.

i2c_slave_address_t address1

Alternate slave 7-bit address and disable.

i2c_slave_address_t address2

Alternate slave 7-bit address and disable.

i2c_slave_address_t address3

Alternate slave 7-bit address and disable.

i2c_slave_address_qual_mode_t qualMode

Qualify mode for slave address 0.

uint8_t qualAddress

Slave address qualifier for address 0.

i2c_slave_bus_speed_t busSpeed

Slave bus speed mode. If the slave function stretches SCL to allow for software response, it must provide sufficient data setup time to the master before releasing the stretched clock. This is accomplished by inserting one clock time of CLKDIV at that point. The *busSpeed* value is used to configure CLKDIV such that one clock time is greater than the tSU;DAT value noted in the I2C bus specification for the I2C mode that is being used. If the *busSpeed* mode is unknown at compile time, use the longest data setup time kI2C_SlaveStandardMode (250 ns)

bool enableSlave

Enable slave mode.

struct _i2c_slave_transfer

#include <fsl_i2c.h> I2C slave transfer structure.

Public Members

i2c_slave_handle_t *handle
Pointer to handle that contains this transfer.

i2c_slave_transfer_event_t event
Reason the callback is being invoked.

uint8_t receivedAddress
Matching address send by master. 7-bits plus R/nW bit0

uint32_t eventMask
Mask of enabled events.

uint8_t *rxData
Transfer buffer for receive data

const uint8_t *txData
Transfer buffer for transmit data

size_t txSize
Transfer size

size_t rxSize
Transfer size

size_t transferredCount
Number of bytes transferred during this transfer.

status_t completionStatus
Success or error code describing how the transfer completed. Only applies for *kI2C_SlaveCompletionEvent*.

struct __i2c_slave_handle
#include <fsl_i2c.h> I2C slave handle structure.

Note: The contents of this structure are private and subject to change.

Public Members

volatile *i2c_slave_transfer_t* transfer
I2C slave transfer.

volatile bool isBusy
Whether transfer is busy.

volatile *i2c_slave_fsm_t* slaveFsm
slave transfer state machine.

i2c_slave_transfer_callback_t callback
Callback function called at transfer event.

void *userData
Callback parameter passed to callback.

2.29 I2S: I2S Driver

2.30 I2S_BRIDGE: I2S bridging and signal sharing configuration

enum _i2s_bridge_share_set_index
I2S Bridge share set.
Values:

enumerator kI2S_BRIDGE_OriginalSignal
Original FLEXCOMM I2S signals

enumerator kI2S_BRIDGE_ShareSet0
share set 0 signals

enumerator kI2S_BRIDGE_ShareSet1
share set 1 signals

enum _i2s_bridge_signal
I2S signal.
Values:

enumerator kI2S_BRIDGE_SignalSCK
SCK signal

enumerator kI2S_BRIDGE_SignalWS
WS signal

enumerator kI2S_BRIDGE_SignalDataIn
Data in signal

enumerator kI2S_BRIDGE_SignalDataOut
Data out signal

enum _i2s_bridge_share_src
I2S signal source.
Values:

enumerator kI2S_BRIDGE_Flexcomm0
Shared signal comes from FLEXCOMM0

enumerator kI2S_BRIDGE_Flexcomm1
Shared signal comes from FLEXCOMM1

enumerator kI2S_BRIDGE_Flexcomm2
Shared signal comes from FLEXCOMM2

enumerator kI2S_BRIDGE_Flexcomm3
Shared signal comes from FLEXCOMM3

enumerator kI2S_BRIDGE_Flexcomm4
Shared signal comes from FLEXCOMM4

enumerator kI2S_BRIDGE_Flexcomm5
Shared signal comes from FLEXCOMM5

enumerator kI2S_BRIDGE_Flexcomm6
Shared signal comes from FLEXCOMM6

enumerator kI2S_BRIDGE_Flexcomm7
Shared signal comes from FLEXCOMM7

enum _i2s_bridge_dataout_mask
I2S Bridge shared data out mask.
Values:

enumerator kI2S_BRIDGE_Flexcomm0DataOut
FLEXCOMM0 DATAOUT Output Enable

enumerator kI2S_BRIDGE_Flexcomm1DataOut
FLEXCOMM1 DATAOUT Output Enable

enumerator kI2S_BRIDGE_Flexcomm2DataOut
FLEXCOMM2 DATAOUT Output Enable

enumerator kI2S_BRIDGE_Flexcomm3DataOut
FLEXCOMM3 DATAOUT Output Enable

enumerator kI2S_BRIDGE_Flexcomm4DataOut
FLEXCOMM4 DATAOUT Output Enable

enumerator kI2S_BRIDGE_Flexcomm5DataOut
FLEXCOMM5 DATAOUT Output Enable

enumerator kI2S_BRIDGE_Flexcomm6DataOut
FLEXCOMM6 DATAOUT Output Enable

enumerator kI2S_BRIDGE_Flexcomm7DataOut

FLEXCOMM7 DATAOUT Output Enable

typedef enum *_i2s_bridge_signal* i2s_bridge_signal_t

I2S signal.

FSL_I2S_BRIDGE_DRIVER_VERSION

Group I2S Bridge driver version for SDK.

Version 2.0.0.

void I2S_BRIDGE_SetFlexcommShareSet(uint32_t flexCommIndex, uint32_t sckSet, uint32_t wsSet, uint32_t dataInSet, uint32_t dataOutSet)

I2S Bridge share set selection for flexcomm instance.

Parameters

- flexCommIndex – index of flexcomm, refer to RM for supported FLEXCOMM instances.
- sckSet – share set for sck, refer to _i2s_bridge_share_set_index
- wsSet – share set for ws, refer to _i2s_bridge_share_set_index
- dataInSet – share set for data in, refer to _i2s_bridge_share_set_index
- dataOutSet – share set for data out, refer to _i2s_bridge_share_set_index

void I2S_BRIDGE_SetFlexcommSignalShareSet(uint32_t flexCommIndex, *i2s_bridge_signal_t* signal, uint32_t set)

I2S Bridge share set selection for a separate signal.

Parameters

- flexCommIndex – index of flexcomm, refer to RM for supported FLEXCOMM instances.
- signal – The signal need to be configured.
- set – share set for the signal, refer to *_i2s_bridge_share_set_index*

void I2S_BRIDGE_SetShareSetSrc(uint32_t setIndex, uint32_t sckShareSrc, uint32_t wsShareSrc, uint32_t dataInShareSrc, uint32_t dataOutShareSrc)

I2S Bridge share set source configure.

Parameters

- setIndex – index of share set, refer *_i2s_bridge_share_set_index*
- sckShareSrc – sck source for this share set, refer to _i2s_bridge_share_src
- wsShareSrc – ws source for this share set, refer to _i2s_bridge_share_src
- dataInShareSrc – data in source for this share set, refer to _i2s_bridge_share_src
- dataOutShareSrc – data out source for this share set, refer to _i2s_bridge_dataout_mask

void I2S_BRIDGE_SetShareSignalSrc(uint32_t setIndex, *i2s_bridge_signal_t* signal, uint32_t shareSrc)

I2S Bridge shared signal source selection for a share set.

Parameters

- setIndex – index of share set, refer to _i2s_bridge_share_set_index

- signal – the shared signal to be configured
- shareSrc – the signal selection, refer to _i2s_bridge_share_src.

2.31 I2S DMA Driver

void I2S_TxTransferCreateHandleDMA(I2S_Type *base, *i2s_dma_handle_t* *handle, *dma_handle_t* *dmaHandle, *i2s_dma_transfer_callback_t* callback, void *userData)

Initializes handle for transfer of audio data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- dmaHandle – pointer to dma handle structure.
- callback – function to be called back when transfer is done or fails.
- userData – pointer to data passed to callback.

status_t I2S_TxTransferSendDMA(I2S_Type *base, *i2s_dma_handle_t* *handle, *i2s_transfer_t* transfer)

Begins or queue sending of the given data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- transfer – data buffer.

Return values

- kStatus_Success –
- kStatus_I2S_Busy – if all queue slots are occupied with unsent buffers.

void I2S_TransferAbortDMA(I2S_Type *base, *i2s_dma_handle_t* *handle)

Aborts transfer of data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

void I2S_RxTransferCreateHandleDMA(I2S_Type *base, *i2s_dma_handle_t* *handle, *dma_handle_t* *dmaHandle, *i2s_dma_transfer_callback_t* callback, void *userData)

Initializes handle for reception of audio data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- dmaHandle – pointer to dma handle structure.
- callback – function to be called back when transfer is done or fails.
- userData – pointer to data passed to callback.

status_t I2S_RxTransferReceiveDMA(I2S_Type *base, *i2s_dma_handle_t* *handle, *i2s_transfer_t* transfer)

Begins or queue reception of data into given buffer.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- transfer – data buffer.

Return values

- kStatus_Success –
- kStatus_I2S_Busy – if all queue slots are occupied with buffers which are not full.

void I2S_DMACallback(*dma_handle_t* *handle, void *userData, bool transferDone, uint32_t tcds)

Invoked from DMA interrupt handler.

Parameters

- handle – pointer to DMA handle structure.
- userData – argument for user callback.
- transferDone – if transfer was done.
- tcds –

void I2S_TransferInstallLoopDMADescriptorMemory(*i2s_dma_handle_t* *handle, void *dmaDescriptorAddr, size_t dmaDescriptorNum)

Install DMA descriptor memory for loop transfer only.

This function used to register DMA descriptor memory for the i2s loop dma transfer.

It must be callbed before I2S_TransferSendLoopDMA/I2S_TransferReceiveLoopDMA and after I2S_RxTransferCreateHandleDMA/I2S_TxTransferCreateHandleDMA.

User should be take care about the address of DMA descriptor pool which required align with 16BYTE at least.

Parameters

- handle – Pointer to i2s DMA transfer handle.
- dmaDescriptorAddr – DMA descriptor start address.
- dmaDescriptorNum – DMA descriptor number.

status_t I2S_TransferSendLoopDMA(I2S_Type *base, *i2s_dma_handle_t* *handle, *i2s_transfer_t* *xfer, uint32_t loopTransferCount)

Send link transfer data using DMA.

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

This function support loop transfer, such as A->B->...->A, the loop transfer chain will be converted into a chain of descriptor and submit to dma. Application must be aware of that the more counts of the loop transfer, then more DMA descriptor memory required, user can use function I2S_InstallDMADescriptorMemory to register the dma descriptor memory.

As the DMA support maximum 1024 transfer count, so application must be aware of that this transfer function support maximum 1024 samples in each transfer, otherwise assert error or error status will be returned. Once the loop transfer start, application can use function I2S_TransferAbortDMA to stop the loop transfer.

Parameters

- base – I2S peripheral base address.
- handle – Pointer to usart_dma_handle_t structure.
- xfer – I2S DMA transfer structure. See *i2s_transfer_t*.
- loopTransferCount – loop count

Return values

kStatus_Success –

status_t I2S_TransferReceiveLoopDMA(I2S_Type *base, *i2s_dma_handle_t* *handle, *i2s_transfer_t* *xfer, uint32_t loopTransferCount)

Receive link transfer data using DMA.

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

This function support loop transfer, such as A->B->...->A, the loop transfer chain will be converted into a chain of descriptor and submit to dma. Application must be aware of that the more counts of the loop transfer, then more DMA descriptor memory required, user can use function I2S_InstallDMADescriptorMemory to register the dma descriptor memory.

As the DMA support maximum 1024 transfer count, so application must be aware of that this transfer function support maximum 1024 samples in each transfer, otherwise assert error or error status will be returned. Once the loop transfer start, application can use function I2S_TransferAbortDMA to stop the loop transfer.

Parameters

- base – I2S peripheral base address.
- handle – Pointer to usart_dma_handle_t structure.
- xfer – I2S DMA transfer structure. See *i2s_transfer_t*.
- loopTransferCount – loop count

Return values

kStatus_Success –

FSL_I2S_DMA_DRIVER_VERSION

I2S DMA driver version 2.3.3.

typedef struct *_i2s_dma_handle* i2s_dma_handle_t

Members not to be accessed / modified outside of the driver.

typedef void (*i2s_dma_transfer_callback_t)(I2S_Type *base, *i2s_dma_handle_t* *handle, *status_t* completionStatus, void *userData)

Callback function invoked from DMA API on completion.

Param base
I2S base pointer.

Param handle
pointer to I2S transaction.

Param completionStatus
status of the transaction.

Param userData
optional pointer to user arguments data.

struct __i2s__dma__handle

#include <fsl_i2s_dma.h> i2s dma handle

Public Members

uint32_t state

Internal state of I2S DMA transfer

uint8_t bytesPerFrame

bytes per frame

i2s_dma_transfer_callback_t completionCallback

Callback function pointer

void *userData

Application data passed to callback

dma_handle_t *dmaHandle

DMA handle

volatile *i2s_transfer_t* i2sQueue[(4U)]

Transfer queue storing transfer buffers

volatile uint8_t queueUser

Queue index where user's next transfer will be stored

volatile uint8_t queueDriver

Queue index of buffer actually used by the driver

dma_descriptor_t *i2sLoopDMADescriptor

descriptor pool pointer

size_t i2sLoopDMADescriptorNum

number of descriptor in descriptors pool

2.32 I2S Driver

void I2S__TxInit(I2S_Type *base, const *i2s_config_t* *config)

Initializes the FLEXCOMM peripheral for I2S transmit functionality.

Ungates the FLEXCOMM clock and configures the module for I2S transmission using a configuration structure. The configuration structure can be custom filled or set with default values by *I2S_TxGetDefaultConfig()*.

Note: This API should be called at the beginning of the application to use the I2S driver.

Parameters

- base – I2S base pointer.
- config – pointer to I2S configuration structure.

void I2S_RxInit(I2S_Type *base, const *i2s_config_t* *config)

Initializes the FLEXCOMM peripheral for I2S receive functionality.

Ungates the FLEXCOMM clock and configures the module for I2S receive using a configuration structure. The configuration structure can be custom filled or set with default values by *I2S_RxGetDefaultConfig()*.

Note: This API should be called at the beginning of the application to use the I2S driver.

Parameters

- base – I2S base pointer.
- config – pointer to I2S configuration structure.

void I2S_TxGetDefaultConfig(*i2s_config_t* *config)

Sets the I2S Tx configuration structure to default values.

This API initializes the configuration structure for use in *I2S_TxInit()*. The initialized structure can remain unchanged in *I2S_TxInit()*, or it can be modified before calling *I2S_TxInit()*. Example:

```
i2s_config_t config;
I2S_TxGetDefaultConfig(&config);
```

Default values:

```
config->masterSlave = kI2S_MasterSlaveNormalMaster;
config->mode = kI2S_ModeI2sClassic;
config->rightLow = false;
config->leftJust = false;
config->pdmData = false;
config->sckPol = false;
config->wsPol = false;
config->divider = 1;
config->oneChannel = false;
config->dataLength = 16;
config->frameLength = 32;
config->position = 0;
config->watermark = 4;
config->txEmptyZero = true;
config->pack48 = false;
```

Parameters

- config – pointer to I2S configuration structure.

void I2S_RxGetDefaultConfig(*i2s_config_t* *config)

Sets the I2S Rx configuration structure to default values.

This API initializes the configuration structure for use in *I2S_RxInit()*. The initialized structure can remain unchanged in *I2S_RxInit()*, or it can be modified before calling *I2S_RxInit()*. Example:

```
i2s_config_t config;
I2S_RxGetDefaultConfig(&config);
```

Default values:

```
config->masterSlave = kI2S_MasterSlaveNormalSlave;
config->mode = kI2S_ModeI2sClassic;
config->rightLow = false;
config->leftJust = false;
config->pdmData = false;
config->sckPol = false;
config->wsPol = false;
config->divider = 1;
config->oneChannel = false;
config->dataLength = 16;
config->frameLength = 32;
config->position = 0;
config->watermark = 4;
config->txEmptyZero = false;
config->pack48 = false;
```

Parameters

- config – pointer to I2S configuration structure.

void I2S_Deinit(I2S_Type *base)

De-initializes the I2S peripheral.

This API gates the FLEXCOMM clock. The I2S module can't operate unless I2S_TxInit or I2S_RxInit is called to enable the clock.

Parameters

- base – I2S base pointer.

void I2S_SetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Transmitter/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 I2S_TxTransferCreateHandle(I2S_Type *base, *i2s_handle_t* *handle, *i2s_transfer_callback_t* callback, void *userData)

Initializes handle for transfer of audio data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- callback – function to be called back when transfer is done or fails.
- userData – pointer to data passed to callback.

status_t I2S_TxTransferNonBlocking(I2S_Type *base, *i2s_handle_t* *handle, *i2s_transfer_t* transfer)

Begins or queue sending of the given data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- transfer – data buffer.

Return values

- kStatus_Success –
- kStatus_I2S_Busy – if all queue slots are occupied with unsent buffers.

void I2S_TxTransferAbort(I2S_Type *base, *i2s_handle_t* *handle)

Aborts sending of data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

void I2S_RxTransferCreateHandle(I2S_Type *base, *i2s_handle_t* *handle, *i2s_transfer_callback_t* callback, void *userData)

Initializes handle for reception of audio data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- callback – function to be called back when transfer is done or fails.
- userData – pointer to data passed to callback.

status_t I2S_RxTransferNonBlocking(I2S_Type *base, *i2s_handle_t* *handle, *i2s_transfer_t* transfer)

Begins or queue reception of data into given buffer.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- transfer – data buffer.

Return values

- kStatus_Success –
- kStatus_I2S_Busy – if all queue slots are occupied with buffers which are not full.

void I2S_RxTransferAbort(I2S_Type *base, *i2s_handle_t* *handle)

Aborts receiving of data.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

status_t I2S_TransferGetCount(I2S_Type *base, *i2s_handle_t* *handle, size_t *count)

Returns number of bytes transferred so far.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

- count – **[out]** number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – there is no non-blocking transaction currently in progress.

status_t I2S_TransferGetErrorCount(I2S_Type *base, *i2s_handle_t* *handle, size_t *count)

Returns number of buffer underruns or overruns.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.
- count – **[out]** number of transmit errors encountered so far by the non-blocking transaction.

Return values

- kStatus_Success –
- kStatus_NoTransferInProgress – there is no non-blocking transaction currently in progress.

static inline void I2S_Enable(I2S_Type *base)

Enables I2S operation.

Parameters

- base – I2S base pointer.

void I2S_EnableSecondaryChannel(I2S_Type *base, uint32_t channel, bool oneChannel, uint32_t position)

Enables I2S secondary channel.

Parameters

- base – I2S base pointer.
- channel – seondary channel channel number, reference _i2s_secondary_channel.
- oneChannel – true is treated as single channel, functionality left channel for this pair.
- position – define the location within the frame of the data, should not bigger than 0x1FFU.

static inline void I2S_DisableSecondaryChannel(I2S_Type *base, uint32_t channel)

Disables I2S secondary channel.

Parameters

- base – I2S base pointer.
- channel – seondary channel channel number, reference _i2s_secondary_channel.

static inline void I2S_Disable(I2S_Type *base)

Disables I2S operation.

Parameters

- base – I2S base pointer.

static inline void I2S_EnableInterrupts(I2S_Type *base, uint32_t interruptMask)

Enables I2S FIFO interrupts.

Parameters

- base – I2S base pointer.
- interruptMask – bit mask of interrupts to enable. See *i2s_flags_t* for the set of constants that should be OR'd together to form the bit mask.

static inline void I2S_DisableInterrupts(I2S_Type *base, uint32_t interruptMask)

Disables I2S FIFO interrupts.

Parameters

- base – I2S base pointer.
- interruptMask – bit mask of interrupts to enable. See *i2s_flags_t* for the set of constants that should be OR'd together to form the bit mask.

static inline uint32_t I2S_GetEnabledInterrupts(I2S_Type *base)

Returns the set of currently enabled I2S FIFO interrupts.

Parameters

- base – I2S base pointer.

Returns

A bitmask composed of *i2s_flags_t* enumerators OR'd together to indicate the set of enabled interrupts.

status_t I2S_EmptyTxFifo(I2S_Type *base)

Flush the valid data in TX fifo.

Parameters

- base – I2S base pointer.

Returns

kStatus_Fail empty TX fifo failed, kStatus_Success empty tx fifo success.

void I2S_TxHandleIRQ(I2S_Type *base, *i2s_handle_t* *handle)

Invoked from interrupt handler when transmit FIFO level decreases.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

void I2S_RxHandleIRQ(I2S_Type *base, *i2s_handle_t* *handle)

Invoked from interrupt handler when receive FIFO level decreases.

Parameters

- base – I2S base pointer.
- handle – pointer to handle structure.

FSL_I2S_DRIVER_VERSION

I2S driver version 2.3.2.

_i2s_status I2S status codes.

Values:

enumerator kStatus_I2S_BufferComplete
Transfer from/into a single buffer has completed

enumerator kStatus_I2S_Done
All buffers transfers have completed

enumerator kStatus_I2S_Busy
Already performing a transfer and cannot queue another buffer

enum _i2s_flags
I2S flags.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kI2S_TxErrorFlag
TX error interrupt

enumerator kI2S_TxLevelFlag
TX level interrupt

enumerator kI2S_RxErrorFlag
RX error interrupt

enumerator kI2S_RxLevelFlag
RX level interrupt

enum _i2s_master_slave
Master / slave mode.
Values:

enumerator kI2S_MasterSlaveNormalSlave
Normal slave

enumerator kI2S_MasterSlaveWsSyncMaster
WS synchronized master

enumerator kI2S_MasterSlaveExtSckMaster
Master using existing SCK

enumerator kI2S_MasterSlaveNormalMaster
Normal master

enum _i2s_mode
I2S mode.
Values:

enumerator kI2S_ModeI2sClassic
I2S classic mode

enumerator kI2S_ModeDspWs50
DSP mode, WS having 50% duty cycle

enumerator kI2S_ModeDspWsShort
DSP mode, WS having one clock long pulse

enumerator kI2S_ModeDspWsLong
DSP mode, WS having one data slot long pulse

_i2s_secondary_channel I2S secondary channel.

Values:

enumerator kI2S_SecondaryChannel1
secondary channel 1

enumerator kI2S_SecondaryChannel2
secondary channel 2

enumerator kI2S_SecondaryChannel3
secondary channel 3

typedef enum *_i2s_flags* i2s_flags_t
I2S flags.

Note: These enums are meant to be OR'd together to form a bit mask.

typedef enum *_i2s_master_slave* i2s_master_slave_t
Master / slave mode.

typedef enum *_i2s_mode* i2s_mode_t
I2S mode.

typedef struct *_i2s_config* i2s_config_t
I2S configuration structure.

typedef struct *_i2s_transfer* i2s_transfer_t
Buffer to transfer from or receive audio data into.

typedef struct *_i2s_handle* i2s_handle_t
Transactional state of the intialized transfer or receive I2S operation.

typedef void (*i2s_transfer_callback_t)(I2S_Type *base, *i2s_handle_t* *handle, *status_t* completionStatus, void *userData)
Callback function invoked from transactional API on completion of a single buffer transfer.

Param base
I2S base pointer.

Param handle
pointer to I2S transaction.

Param completionStatus
status of the transaction.

Param userData
optional pointer to user arguments data.

I2S_NUM_BUFFERS
Number of buffers .

struct _i2s_config
#include <fsl_i2s.h> I2S configuration structure.

Public Members

i2s_master_slave_t masterSlave
Master / slave configuration

i2s_mode_t mode
I2S mode

bool rightLow
Right channel data in low portion of FIFO

bool leftJust
Left justify data in FIFO

bool pdmData
Data source is the D-Mic subsystem

bool sckPol
SCK polarity

bool wsPol
WS polarity

uint16_t divider
Flexcomm function clock divider (1 - 4096)

bool oneChannel
true mono, false stereo

uint8_t dataLength
Data length (4 - 32)

uint16_t frameLength
: Frame width (4 - 512)

uint16_t position
: Data position in the frame

uint8_t watermark
: FIFO trigger level

bool txEmptyZero
: Transmit zero when buffer becomes empty or last item

bool pack48
: Packing format for 48-bit data (false - 24 bit values, true - alternating 32-bit and 16-bit values)

struct __i2s_transfer
: *#include <fsl_i2s.h>* Buffer to transfer from or receive audio data into.

Public Members

uint8_t *data
: Pointer to data buffer.

size_t dataSize
: Buffer size in bytes.

struct __i2s_handle
: *#include <fsl_i2s.h>* Members not to be accessed / modified outside of the driver.

Public Members

volatile uint32_t state
: State of transfer

i2s_transfer_callback_t completionCallback
: Callback function pointer

void *userData
: Application data passed to callback

bool oneChannel
: true mono, false stereo

uint8_t dataLength
: Data length (4 - 32)

bool pack48
: Packing format for 48-bit data (false - 24 bit values, true - alternating 32-bit and 16-bit values)

uint8_t watermark
: FIFO trigger level

bool useFifo48H
: When dataLength 17-24: true use FIFOWR48H, false use FIFOWR

volatile *i2s_transfer_t* i2sQueue[(4U)]
: Transfer queue storing transfer buffers

volatile uint8_t queueUser
: Queue index where user's next transfer will be stored

volatile uint8_t queueDriver
: Queue index of buffer actually used by the driver

volatile uint32_t errorCount
: Number of buffer underruns/overruns

volatile uint32_t transferCount
: Number of bytes transferred

2.33 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.

2.34 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;
config->hdrMode             = (uint8_t)kI3C_HDRModeDDR;
config->nakAllRequest       = false;
config->ignoreS0S1Error     = false;
```

(continues on next page)

(continued from previous page)

```
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.

2.35 I3C Master DMA Driver

typedef struct *_i3c_master_dma_handle* i3c_master_dma_handle_t

typedef struct *_i3c_master_dma_callback* i3c_master_dma_callback_t

i3c master callback functions.

void I3C_MasterTransferCreateHandleDMA(I3C_Type *base, *i3c_master_dma_handle_t* *handle, const *i3c_master_dma_callback_t* *callback, void *userData, *dma_handle_t* *rxDmaHandle, *dma_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_MasterTransferDMA(I3C_Type *base, *i3c_master_dma_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.

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_MasterTransferGetCountDMA(I3C_Type *base, *i3c_master_dma_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_MasterTransferAbortDMA(I3C_Type *base, *i3c_master_dma_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_MasterTransferDMAHandleIRQ(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.
- handle – Pointer to the I3C master DMA driver handle.

void (*slave2Master)(I3C_Type *base, void *userData)

Transfer complete callback

void (*ibiCallback)(I3C_Type *base, *i3c_master_dma_handle_t* *handle, *i3c_ibi_type_t* ibiType, *i3c_ibi_state_t* ibiState)

IBI event callback

void (*transferComplete)(I3C_Type *base, *i3c_master_dma_handle_t* *handle, *status_t* status, void *userData)

Transfer complete callback

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_dma_callback_t callback
: Callback function pointer.

void *userData
: Application data passed to callback.

dma_handle_t *rxDmaHandle
: Handle for receive DMA channel.

dma_handle_t *txDmaHandle
: Handle for transmit DMA channel.

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.

uint32_t transDataSize
: Transferred data size.

uint8_t workaroundBuff[16]
: Workaround buffer to store temporary data.

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*.

I3C_Type *base
: I3C base pointer.

i3c_slave_dma_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.

i3c_slave_dma_callback_t callback
: Callback function called at transfer event.

dma_handle_t *rxDmaHandle
: Handle for receive DMA channel.

dma_handle_t *txDmaHandle
: Handle for transmit DMA channel.

void *userData
: Callback parameter passed to callback.

struct __i3c_master_dma_callback
: *#include <fsl_i3c_dma.h>* i3c master callback functions.

struct __i3c_master_dma_handle
: *#include <fsl_i3c_dma.h>* Driver handle for master DMA APIs.

Note: The contents of this structure are private and subject to change.

2.36 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.

2.37 I3C Slave DMA Driver

void I3C_SlaveTransferCreateHandleDMA(I3C_Type *base, *i3c_slave_dma_handle_t* *handle, *i3c_slave_dma_callback_t* callback, void *userData, *dma_handle_t* *rxDmaHandle, *dma_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_SlaveTransferDMA(I3C_Type *base, *i3c_slave_dma_handle_t* *handle, *i3c_slave_dma_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.

void I3C_SlaveTransferAbortDMA(I3C_Type *base, *i3c_slave_dma_handle_t* *handle)

Abort a slave dma non-blocking transfer in a early time.

Parameters

- base – I3C peripheral base address
- handle – pointer to i3c_slave_dma_handle_t structure

void I3C_SlaveTransferDMAHandleIRQ(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.
- handle – Pointer to the I3C slave DMA driver handle.

typedef struct *_i3c_slave_dma_handle* i3c_slave_dma_handle_t

typedef struct *_i3c_slave_dma_transfer* i3c_slave_dma_transfer_t

I3C slave transfer structure.

typedef void (*i3c_slave_dma_callback_t)(I3C_Type *base, *i3c_slave_dma_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_dma_transfer

#include <fsl_i3c_dma.h> I3C slave transfer structure.

struct _i3c_slave_dma_handle

#include <fsl_i3c_dma.h> I3C slave dma handle structure.

Note: The contents of this structure are private and subject to change.

2.38 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

2.39 IAP Boot Driver

typedef struct *_iap_boot_option* iap_boot_option_t
IAP boot option.

void IAP_RunBootLoader(*iap_boot_option_t* *option)
Invoke into ROM with specified boot parameters.

Parameters

- option – Boot parameters. Refer to *iap_boot_option_t*.

IAP_BOOT_OPTION_TAG
IAP boot option tag

IAP_BOOT_OPTION_MODE_MASTER
IAP boot option mode

IAP_BOOT_OPTION_MODE_ISP

struct _iap_boot_option
#include <fsl_iap.h> IAP boot option.

union option

Public Members

struct *_iap_boot_option* B

uint32_t U

struct B

Public Members

uint32_t bootImageIndex
reserved field.

uint32_t instance
FlexSPI boot image index for FlexSPI NOR flash.

uint32_t bootInterface
Only used when boot interface is FlexSPI/SD/MMC.

uint32_t mode
RT500: 0: USART 2: SPI 3: USB HID 4:FlexSPI 6:SD 7:MMC. RT600: 0: USART 1: I2C 2: SPI 3: USB HID 4:FlexSPI 7:SD 8:MMC

2.40 IAP: In Application Programming Driver

FSL_IAP_DRIVER_VERSION
IAP driver version.

2.41 IAP FlexSPI Driver

FlexSPI Driver status group.
Values:

enumerator kStatusGroup_FlexSPI

enumerator kStatusGroup_FlexSPINOR

enum _flexspi_status
FlexSPI Driver status.
Values:

enumerator kStatus_FLEXSPI_Success
API is executed successfully

enumerator kStatus_FLEXSPI_Fail
API is executed fails

enumerator kStatus_FLEXSPI_InvalidArgument
Invalid argument

enumerator kStatus_FLEXSPI_SequenceExecutionTimeout
The FlexSPI Sequence Execution timeout

enumerator kStatus_FLEXSPI_InvalidSequence
The FlexSPI LUT sequence invalid

enumerator kStatus_FLEXSPI_DeviceTimeout
The FlexSPI device timeout

enumerator kStatus_FLEXSPINOR_ProgramFail
Status for Page programming failure

enumerator kStatus_FLEXSPINOR_EraseSectorFail
Status for Sector Erase failure

enumerator kStatus_FLEXSPINOR_EraseAllFail
Status for Chip Erase failure

enumerator kStatus_FLEXSPINOR_WaitTimeout
Status for timeout

enumerator kStatus_FLEXSPINOR_NotSupported

enumerator kStatus_FLEXSPINOR_WriteAlignmentError
Status for Alignement error

enumerator kStatus_FLEXSPINOR_CommandFailure
Status for Erase/Program Verify Error

enumerator kStatus_FLEXSPINOR_SFDP_NotFound
Status for SFDP read failure

enumerator kStatus_FLEXSPINOR_Unsupported_SFDP_Version
Status for Unrecognized SFDP version

enumerator kStatus_FLEXSPINOR_Flash_NotFound
Status for Flash detection failure

enumerator kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed
Status for DDR Read dummy probe failure

Flash Configuration Option0 device_type.

Values:

enumerator kSerialNorCfgOption_Tag

enumerator kSerialNorCfgOption_DeviceType_ReadSFDP_SDR

enumerator kSerialNorCfgOption_DeviceType_ReadSFDP_DDR

enumerator kSerialNorCfgOption_DeviceType_HyperFLASH1V8

enumerator kSerialNorCfgOption_DeviceType_HyperFLASH3V0

enumerator kSerialNorCfgOption_DeviceType_MacronixOctalDDR

enumerator kSerialNorCfgOption_DeviceType_MacronixOctalSDR

enumerator kSerialNorCfgOption_DeviceType_MicronOctalDDR

enumerator kSerialNorCfgOption_DeviceType_MicronOctalSDR

enumerator kSerialNorCfgOption_DeviceType_AdestoOctalDDR

enumerator kSerialNorCfgOption_DeviceType_AdestoOctalSDR

Flash Configuration Option0 quad_mode_setting.
Values:

enumerator kSerialNorQuadMode_NotConfig

enumerator kSerialNorQuadMode_StatusReg1_Bit6

enumerator kSerialNorQuadMode_StatusReg2_Bit1

enumerator kSerialNorQuadMode_StatusReg2_Bit7

enumerator kSerialNorQuadMode_StatusReg2_Bit1_0x31

Flash Configuration Option0 misc_mode.
Values:

enumerator kSerialNorEnhanceMode_Disabled

enumerator kSerialNorEnhanceMode_0_4_4_Mode

enumerator kSerialNorEnhanceMode_0_8_8_Mode

enumerator kSerialNorEnhanceMode_DataOrderSwapped

enumerator kSerialNorEnhanceMode_2ndPinMux

FLEXSPI_RESET_PIN boot configurations in OTP.
Values:

enumerator kFlashResetLogic_Disabled

enumerator kFlashResetLogic_ResetPin

enumerator kFlashResetLogic_JedecHwReset

Flash Configuration Option1 flash_connection.
Values:

enumerator kSerialNorConnection_SinglePortA

enumerator kSerialNorConnection_Parallel

enumerator kSerialNorConnection_SinglePortB

enumerator kSerialNorConnection_BothPorts

Flash Device Mode Configuration Sequence.
Values:

enumerator kRestoreSequence_None

enumerator kRestoreSequence_HW_Reset

enumerator kRestoreSequence_4QPI_FF

enumerator kRestoreSequence_5QPI_FF

enumerator kRestoreSequence_8QPI_FF

enumerator kRestoreSequence_Send_F0

enumerator kRestoreSequence_Send_66_99

enumerator kRestoreSequence_Send_6699_9966

enumerator kRestoreSequence_Send_06_FF

Flash Config Mode Definition.

Values:

enumerator kFlashInstMode_ExtendedSpi

enumerator kFlashInstMode_0_4_4_SDR

enumerator kFlashInstMode_0_4_4_DDR

enumerator kFlashInstMode_QPI_SDR

enumerator kFlashInstMode_QPI_DDR

enumerator kFlashInstMode_OPI_SDR

enumerator kFlashInstMode_OPI_DDR

Flash Device Type Definition.

Values:

enumerator kFlexSpiDeviceType_SerialNOR
Flash devices are Serial NOR

enumerator kFlexSpiDeviceType_SerialNAND
Flash devices are Serial NAND

enumerator kFlexSpiDeviceType_SerialRAM
Flash devices are Serial RAM/HyperFLASH

enumerator kFlexSpiDeviceType_MCP_NOR_NAND
Flash device is MCP device, A1 is Serial NOR, A2 is Serial NAND

enumerator kFlexSpiDeviceType_MCP_NOR_RAM
Flash deivce is MCP device, A1 is Serial NOR, A2 is Serial RAMs

Flash Pad Definitions.

Values:

enumerator kSerialFlash_1Pad

enumerator kSerialFlash_2Pads

enumerator kSerialFlash_4Pads

enumerator kSerialFlash_8Pads

Flash Configuration Command Type.

Values:

enumerator kDeviceConfigCmdType_Generic
Generic command, for example: configure dummy cycles, drive strength, etc

enumerator kDeviceConfigCmdType_QuadEnable
Quad Enable command

enumerator kDeviceConfigCmdType_Spi2Xpi
Switch from SPI to DPI/QPI/OPI mode

enumerator kDeviceConfigCmdType_Xpi2Spi
Switch from DPI/QPI/OPI to SPI mode

enumerator kDeviceConfigCmdType_Spi2NoCmd
Switch to 0-4-4/0-8-8 mode

enumerator kDeviceConfigCmdType_Reset
Reset device command

enum _FlexSPIOperationType
FlexSPI Operation Type.

Values:

enumerator kFlexSpiOperation_Command
FlexSPI operation: Only command, both TX and

enumerator kFlexSpiOperation_Config
RX buffer are ignored. FlexSPI operation: Configure device mode, the

enumerator kFlexSpiOperation_Write
TX FIFO size is fixed in LUT. FlexSPI operation: Write, only TX buffer is

enumerator kFlexSpiOperation_Read
effective FlexSPI operation: Read, only Rx Buffer is

enumerator kFlexSpiOperation_End
effective.

typedef struct *_serial_nor_config_option* serial_nor_config_option_t
Serial NOR Configuration Option.

typedef struct *_lut_sequence* flexspi_lut_seq_t
FlexSPI LUT Sequence structure.

typedef struct *_FlexSPIConfig* flexspi_mem_config_block_t

FlexSPI Memory Configuration Block.

typedef enum *_FlexSPIOperationType* flexspi_operation_t

FlexSPI Operation Type.

typedef struct *_FlexSpiXfer* flexspi_xfer_t

FlexSPI Transfer Context.

typedef struct *_flexspi_nor_config* flexspi_nor_config_t

Serial NOR configuration block.

status_t IAP_FlexspiNorInit(uint32_t instance, *flexspi_nor_config_t* *config)

Initialize Serial NOR devices via FlexSPI.

This function configures the FlexSPI controller with the arguments pointed by param config.

Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi_nor_config_t*.

Returns

The status flags. This is a member of the enumeration *_flexspi_status*

status_t IAP_FlexspiNorPageProgram(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t dstAddr, const uint32_t *src)

Program data to Serial NOR via FlexSPI.

This function Program data to specified destination address.

Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi_nor_config_t*.
- dstAddr – The destination address to be programmed.
- src – Points to the buffer which hold the data to be programmed.

Returns

The status flags. This is a member of the enumeration *_flexspi_status*

status_t IAP_FlexspiNorEraseAll(uint32_t instance, *flexspi_nor_config_t* *config)

Erase all the Serial NOR devices connected on FlexSPI.

Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi_nor_config_t*.

Returns

The status flags. This is a member of the enumeration *_flexspi_status*

status_t IAP_FlexspiNorErase(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t start, uint32_t length)

Erase Flash Region specified by address and length.

Parameters

- instance – FlexSPI controller instance, only support 0.

- config – The Flash configuration block. Refer to *flexspi_nor_config_t*.
- start – The start address to be erased.
- length – The length to be erased.

Returns
The status flags. This is a member of the enumeration *_flexspi_status*

status_t IAP_FlexspiNorEraseSector(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t address)

Erase one sector specified by address.

Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi_nor_config_t*.
- address – The address of the sector to be erased.

Returns
The status flags. This is a member of the enumeration *_flexspi_status*

status_t IAP_FlexspiNorEraseBlock(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t address)

Erase one block specified by address.

Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi_nor_config_t*.
- address – The address of the block to be erased.

Returns
The status flags. This is a member of the enumeration *_flexspi_status*

status_t IAP_FlexspiNorGetConfig(uint32_t instance, *flexspi_nor_config_t* *config, *serial_nor_config_option_t* *option)

Get FlexSPI NOR Configuration Block based on specified option.

Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi_nor_config_t*.
- option – The Flash Configuration Option block. Refer to *serial_nor_config_option_t*.

Returns
The status flags. This is a member of the enumeration *_flexspi_status*

status_t IAP_FlexspiNorRead(uint32_t instance, *flexspi_nor_config_t* *config, uint32_t *dst, uint32_t start, uint32_t bytes)

Read data from Flexspi NOR Flash.

Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi_nor_config_t*.
- dst – Buffer address used to store the read data.
- start – The Read address.
- bytes – The Read size

Returns
The status flags. This is a member of the enumeration *_flexspi_status*

status_t IAP_FlexspiXfer(uint32_t instance, *flexspi_xfer_t* *xfer)

Get FlexSPI Xfer data.

Parameters

- instance – FlexSPI controller instance, only support 0.
- xfer – The FlexSPI Transfer Context block. Refer to *flexspi_xfer_t*.

Returns
The status flags. This is a member of the enumeration *_flexspi_status*

status_t IAP_FlexspiUpdateLut(uint32_t instance, uint32_t seqIndex, const uint32_t *lutBase, uint32_t numberOfSeq)

Update FlexSPI Lookup table.

Parameters

- instance – FlexSPI controller instance, only support 0.
- seqIndex – The index of FlexSPI LUT to be updated.
- lutBase – Points to the buffer which hold the LUT data to be programmed.
- numberOfSeq – The number of LUT seq that need to be updated.

Returns
The status flags. This is a member of the enumeration *_flexspi_status*

status_t IAP_FlexspiSetClockSource(uint32_t clockSrc)

Set the clock source for FlexSPI.

Parameters

- clockSrc – Clock source for flexspi interface.

Returns
The status flags. This is a member of the enumeration *_flexspi_status*

void IAP_FlexspiConfigClock(uint32_t instance, uint32_t freqOption, uint32_t sampleClkMode)

Configure the flexspi interface clock frequency and data sample mode.

Parameters

- instance – FlexSPI controller instance, only support 0.
- freqOption – FlexSPI interface clock frequency selection.
- sampleClkMode – FlexSPI controller data sample mode.

Returns
The status flags. This is a member of the enumeration *_flexspi_status*

status_t IAP_FlexspiNorAutoConfig(uint32_t instance, *flexspi_nor_config_t* *config, *serial_nor_config_option_t* *option)

Configure flexspi nor automatically.

Parameters

- instance – FlexSPI controller instance, only support 0.
- config – The Flash configuration block. Refer to *flexspi_nor_config_t*.
- option – The Flash Configuration Option block. Refer to *serial_nor_config_option_t*.

Returns
The status flags. This is a member of the enumeration *_flexspi_status*

NOR_CMD_INDEX_READ
FlexSPI LUT command.
0

NOR_CMD_INDEX_READSTATUS
1

NOR_CMD_INDEX_WRITEENABLE
2

NOR_CMD_INDEX_ERASESECTOR
3

NOR_CMD_INDEX_PAGEPROGRAM
4

NOR_CMD_INDEX_CHIPERASE
5

NOR_CMD_INDEX_DUMMY
6

NOR_CMD_INDEX_ERASEBLOCK
7

NOR_CMD_LUT_SEQ_IDX_READ
0 READ LUT sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READSTATUS
1 Read Status LUT sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_READSTATUS_XPI
2 Read status DPI/QPI/OPI sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_WRITEENABLE
3 Write Enable sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_WRITEENABLE_XPI
4 Write Enable DPI/QPI/OPI sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_ERASESECTOR
5 Erase Sector sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_ERASEBLOCK
8 Erase Block sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM
9 Program sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_CHIPERASE
11 Chip Erase sequence in lookupTable id stored in config block

NOR_CMD_LUT_SEQ_IDX_READ_SFDP
13 Read SFDP sequence in lookupTable id stored in config block

NOR_CMD_LUT_SEQ_IDX_RESTORE_NOCMD
14 Restore 0-4-4/0-8-8 mode sequence id in lookupTable stored in config block

NOR_CMD_LUT_SEQ_IDX_EXIT_NOCMD
15 Exit 0-4-4/0-8-8 mode sequence id in lookupTable stored in config blobk

struct _serial_nor_config_option
#include <fsl_iap.h> Serial NOR Configuration Option.

union flash_run_context_t
#include <fsl_iap.h> Flash Run Context.

Public Members

struct *flash_run_context_t* B

uint32_t U

struct _lut_sequence
#include <fsl_iap.h> FlexSPI LUT Sequence structure.

Public Members

uint8_t seqNum
Sequence Number, valid number: 1-16

uint8_t seqId
Sequence Index, valid number: 0-15

struct flexspi_dll_time_t
#include <fsl_iap.h> FlexSPI Dll Time Block.

struct _FlexSPIConfig
#include <fsl_iap.h> FlexSPI Memory Configuration Block.

Public Members

uint32_t tag
[0x000-0x003] Tag, fixed value 0x42464346UL

uint32_t version
: [0x004-0x007] Version,[31:24] -'V', [23:16] - Major, [15:8] - Minor, [7:0] - bugfix

uint32_t reserved0
: [0x008-0x00b] Reserved for future use

uint8_t readSampleClkSrc
: [0x00c-0x00c] Read Sample Clock Source, valid value: 0/1/3

uint8_t csHoldTime
: [0x00d-0x00d] CS hold time, default value: 3

uint8_t csSetupTime
: [0x00e-0x00e] CS setup time, default value: 3

uint8_t columnAddressWidth
: [0x00f-0x00f] Column Address with, for HyperBus protocol, it is fixed to 3, For

uint8_t deviceModeCfgEnable
: Serial NAND, need to refer to datasheet [0x010-0x010] Device Mode Configure enable flag, 1 - Enable, 0 - Disable

uint8_t deviceModeType
: [0x011-0x011] Specify the configuration command type:Quad Enable, DPI/QPI/OPI switch,

uint16_t waitTimeCfgCommands
: Generic configuration, etc. [0x012-0x013] Wait time for all configuration commands, unit: 100us, Used for

flexspi_lut_seq_t deviceModeSeq
: DPI/QPI/OPI switch or reset command [0x014-0x017] Device mode sequence info, [7:0] - LUT sequence id, [15:8] - LUt

uint32_t deviceModeArg
: sequence number, [31:16] Reserved [0x018-0x01b] Argument/Parameter for device configuration

uint8_t configCmdEnable
: [0x01c-0x01c] Configure command Enable Flag, 1 - Enable, 0 - Disable

uint8_t configModeType[3]
: [0x01d-0x01f] Configure Mode Type, similar as deviceModeTpe

flexspi_lut_seq_t configCmdSeqs[3]
: [0x020-0x02b] Sequence info for Device Configuration command, similar as deviceModeSeq

uint32_t reserved1
: [0x02c-0x02f] Reserved for future use

uint32_t configCmdArgs[3]

[0x030-0x03b] Arguments/Parameters for device Configuration commands

uint32_t reserved2

[0x03c-0x03f] Reserved for future use

uint32_t controllerMiscOption

[0x040-0x043] Controller Misc Options, see Misc feature bit definitions for more

uint8_t deviceType

details [0x044-0x044] Device Type: See Flash Type Definition for more details

uint8_t sflashPadType

[0x045-0x045] Serial Flash Pad Type: 1 - Single, 2 - Dual, 4 - Quad, 8 - Octal

uint8_t serialClkFreq

[0x046-0x046] Serial Flash Frequencey, device specific definitions, See System Boot

uint8_t lutCustomSeqEnable

Chapter for more details [0x047-0x047] LUT customization Enable, it is required if the program/erase cannot

uint32_t reserved3[2]

be done using 1 LUT sequence, currently, only applicable to HyperFLASH [0x048-0x04f] Reserved for future use

uint32_t sflashA1Size

[0x050-0x053] Size of Flash connected to A1

uint32_t sflashA2Size

[0x054-0x057] Size of Flash connected to A2

uint32_t sflashB1Size

[0x058-0x05b] Size of Flash connected to B1

uint32_t sflashB2Size

[0x05c-0x05f] Size of Flash connected to B2

uint32_t csPadSettingOverride

[0x060-0x063] CS pad setting override value

uint32_t sclkPadSettingOverride

[0x064-0x067] SCK pad setting override value

uint32_t dataPadSettingOverride

[0x068-0x06b] data pad setting override value

uint32_t dqsPadSettingOverride

[0x06c-0x06f] DQS pad setting override value

uint32_t timeoutInMs
: [0x070-0x073] Timeout threshold for read status command

uint32_t commandInterval
: [0x074-0x077] CS deselect interval between two commands

flexspi_dll_time_t dataValidTime[2]
: [0x078-0x07b] CLK edge to data valid time for PORT A and PORT B

uint16_t busyOffset
: [0x07c-0x07d] Busy offset, valid value: 0-31

uint16_t busyBitPolarity
: [0x07e-0x07f] Busy flag polarity, 0 - busy flag is 1 when flash device is busy, 1 -

uint32_t lookupTable[64]
: busy flag is 0 when flash device is busy [0x080-0x17f] Lookup table holds Flash command sequences

flexspi_lut_seq_t lutCustomSeq[12]
: [0x180-0x1af] Customizable LUT Sequences

uint32_t reserved4[4]
: [0x1b0-0x1bf] Reserved for future use

struct __FlexSpiXfer
: *#include <fsl_iap.h>* FlexSPI Transfer Context.

Public Members

flexspi_operation_t operation
: FlexSPI operation

uint32_t baseAddress
: FlexSPI operation base address

uint32_t seqId
: Sequence Id

uint32_t seqNum
: Sequence Number

bool isParallelModeEnable
: Is a parallel transfer

uint32_t *txBuffer
: Tx buffer

uint32_t txSize
Tx size in bytes

uint32_t *rxBuffer
Rx buffer

uint32_t rxSize
Rx size in bytes

struct __flexspi_nor_config
#include <fsl_iap.h> Serial NOR configuration block.

Public Members

flexspi_mem_config_block_t memConfig
Common memory configuration info via FlexSPI

uint32_t pageSize
Page size of Serial NOR

uint32_t sectorSize
Sector size of Serial NOR

uint8_t ipcmdSerialClkFreq
Clock frequency for IP command

uint8_t isUniformBlockSize
Sector/Block size is the same

uint8_t isDataOrderSwapped
Data order (D0, D1, D2, D3) is swapped (D1,D0, D3, D2)

uint8_t reserved0[1]
Reserved for future use

uint8_t serialNorType
Serial NOR Flash type: 0/1/2/3

uint8_t needExitNoCmdMode
Need to exit NoCmd mode before other IP command

uint8_t halfClkForNonReadCmd
Half the Serial Clock for non-read command: true/false

uint8_t needRestoreNoCmdMode
Need to Restore NoCmd mode after IP commmand execution

uint32_t blockSize
Block size

uint32_t flashStateCtx
Flash State Context

uint32_t reserve2[10]
Reserved for future use

union option0

Public Members

struct *_serial_nor_config_option* B

uint32_t U

struct B

Public Members

uint32_t max_freq
Maximum supported Frequency

uint32_t misc_mode
miscellaneous mode

uint32_t quad_mode_setting
Quad mode setting

uint32_t cmd_pads
Command pads

uint32_t query_pads
SFDP read pads

uint32_t device_type
Device type

uint32_t option_size
Option size, in terms of uint32_t, size = (option_size + 1) * 4

uint32_t tag
Tag, must be 0x0C

union option1

Public Members

struct *_serial_nor_config_option* B

uint32_t U

struct B

Public Members

uint32_t dummy_cycles
Dummy cycles before read

uint32_t status_override
Override status register value during device mode configuration

uint32_t pinmux_group
The pinmux group selection

uint32_t dqs_pinmux_group
The DQS Pinmux Group Selection

uint32_t drive_strength
The Drive Strength of FlexSPI Pads

uint32_t flash_connection
Flash connection option: 0 - Single Flash connected to port A, 1 -

struct B

2.42 IAP OTP Driver

OTP Status Group.

Values:

enumerator kStatusGroup_OtpGroup

OTP Error Status definitions.

Values:

enumerator kStatus_OTP_InvalidAddress
Invalid OTP address

enumerator kStatus_OTP_ProgramFail
Program Fail

enumerator kStatus_OTP_CrcFail
CrcCheck Fail

enumerator kStatus_OTP_Error
Errors happened during OTP operation

enumerator kStatus_OTP_EccCheckFail
Ecc Check failed during OTP operation

enumerator kStatus_OTP_Locked
OTP Fuse field has been locked

enumerator kStatus_OTP_Timeout
OTP operation time out

enumerator kStatus_OTP_CrcCheckPass
OTP CRC Check Pass

status_t IAP_OtpInit(uint32_t src_clk_freq)
Initialize OTP controller.

This function enables OTP Controller clock.

Parameters

- src_clk_freq – The Frequency of the source clock of OTP controller

Returns
kStatus_Success

status_t IAP_OtpDeinit(void)
De-Initialize OTP controller.

This functin disables OTP Controller Clock.

Returns
kStatus_Success

status_t IAP_OtpFuseRead(uint32_t addr, uint32_t *data)
Read Fuse value from OTP Fuse Block.

This function read fuse data from OTP Fuse block to specified data buffer.

Parameters

- addr – Fuse address
- data – Buffer to hold the data read from OTP Fuse block

Returns
kStatus_Success - Data read from OTP Fuse block successfully kStatus_InvalidArgument - data pointer is invalid kStatus_OTP_EccCheckFail - Ecc Check Failed kStatus_OTP_Error - Other Errors

status_t IAP_OtpFuseProgram(uint32_t addr, uint32_t data, bool lock)
Program value to OTP Fuse block.

This function program data to specified OTP Fuse address.

Parameters

- addr – Fuse address
- data – data to be programmed into OTP Fuse block
- lock – lock the fuse field or not

Returns

kStatus_Success - Data has been programmed into OTP Fuse block successfully kStatus_OTP_ProgramFail - Fuse programming failed kStatus_OTP_Locked - The address to be programmed into is locked kStatus_OTP_Error - Other Errors

status_t IAP_OtpShadowRegisterReload(void)

Reload all shadow registers from OTP fuse block.

This function reloads all the shadow registers from OTP Fuse block

Returns

kStatus_Success - Shadow registers' reloadding succeeded. kStatus_OTP_EccCheckFail - Ecc Check Failed kStatus_OTP_Error - Other Errors

status_t IAP_OtpCrcCheck(uint32_t start_addr, uint32_t end_addr, uint32_t crc_addr)

Do CRC Check via OTP controller.

This function checks whether data in specified fuse address ranges match the crc value in the specified CRC address and return the actual crc value as needed.

Parameters

- start_addr – Start address of selected Fuse address range
- end_addr – End address of selected Fuse address range
- crc_addr – Address that hold CRC data

Returns

kStatus_Success CRC check succeeded, CRC value matched. kStatus_InvalidArgument - Invalid Argument kStatus_OTP_EccCheckFail Ecc Check Failed kStatus_OTP_CrcFail CRC Check Failed

status_t IAP_OtpCrcCalc(uint32_t *src, uint32_t numberOfWords, uint32_t *crcChecksum)

Calculate the CRC checksum for specified data for OTP.

This function calculates the CRC checksum for specified data for OTP

Parameters

- src – the source address of data
- numberOfWords – number of Fuse words
- crcChecksum – Buffer to store the CRC checksum

Returns

kStatus_Success CRC checksum is computed successfully. kStatus_InvalidArgument - Invalid Argument

2.43 INPUTMUX: Input Multiplexing Driver

FSL_INPUTMUX_DRIVER_VERSION

Group interrupt driver version for SDK.

enum _inputmux_connection_t

INPUTMUX connections type.

Values:

enumerator kINPUTMUX_Sct0PinInp0ToSct0

SCT INMUX.

enumerator kINPUTMUX_Sct0PinInp1ToSct0

enumerator kINPUTMUX_Sct0PinInp2ToSct0

enumerator kINPUTMUX_Sct0PinInp3ToSct0

enumerator kINPUTMUX_Sct0PinInp4ToSct0

enumerator kINPUTMUX_Sct0PinInp5ToSct0

enumerator kINPUTMUX_Sct0PinInp6ToSct0

enumerator kINPUTMUX_Sct0PinInp7ToSct0

enumerator kINPUTMUX_Ctimer0Mat0ToSct0

enumerator kINPUTMUX_Ctimer1Mat0ToSct0

enumerator kINPUTMUX_Ctimer2Mat0ToSct0

enumerator kINPUTMUX_Ctimer3Mat0ToSct0

enumerator kINPUTMUX_Ctimer4Mat0ToSct0

enumerator kINPUTMUX_AdcIrqToSct0

enumerator kINPUTMUX_GpioIntBmatchToSct0

enumerator kINPUTMUX_Usb1FrameToggleToSct0

enumerator kINPUTMUX_Cmp0OutToSct0

enumerator kINPUTMUX_SharedI2s0SclkToSct0

enumerator kINPUTMUX_SharedI2s1SclkToSct0

enumerator kINPUTMUX_SharedI2s0WsToSct0

enumerator kINPUTMUX_SharedI2s1WsToSct0

enumerator kINPUTMUX_MclkToSct0

enumerator kINPUTMUX_ArmTxevToSct0

enumerator kINPUTMUX_DebugHaltedToSct0
Pin Interrupt.

enumerator kINPUTMUX_GpioPort0Pin0ToPintsel

enumerator kINPUTMUX_GpioPort0Pin1ToPintsel

enumerator kINPUTMUX_GpioPort0Pin2ToPintsel

enumerator kINPUTMUX_GpioPort0Pin3ToPintsel

enumerator kINPUTMUX_GpioPort0Pin4ToPintsel

enumerator kINPUTMUX_GpioPort0Pin5ToPintsel

enumerator kINPUTMUX_GpioPort0Pin6ToPintsel

enumerator kINPUTMUX_GpioPort0Pin7ToPintsel

enumerator kINPUTMUX_GpioPort0Pin8ToPintsel

enumerator kINPUTMUX_GpioPort0Pin9ToPintsel

enumerator kINPUTMUX_GpioPort0Pin10ToPintsel

enumerator kINPUTMUX_GpioPort0Pin11ToPintsel

enumerator kINPUTMUX_GpioPort0Pin12ToPintsel

enumerator kINPUTMUX_GpioPort0Pin13ToPintsel

enumerator kINPUTMUX_GpioPort0Pin14ToPintsel

enumerator kINPUTMUX_GpioPort0Pin15ToPintsel

enumerator kINPUTMUX_GpioPort0Pin16ToPintsel

enumerator kINPUTMUX_GpioPort0Pin17ToPintsel

enumerator kINPUTMUX_GpioPort0Pin18ToPintsel

enumerator kINPUTMUX_GpioPort0Pin19ToPintsel

enumerator kINPUTMUX_GpioPort0Pin20ToPintsel

enumerator kINPUTMUX_GpioPort0Pin21ToPintsel

enumerator kINPUTMUX_GpioPort0Pin22ToPintsel

enumerator kINPUTMUX_GpioPort0Pin23ToPintsel

enumerator kINPUTMUX_GpioPort0Pin24ToPintsel

enumerator kINPUTMUX_GpioPort0Pin25ToPintsel

enumerator kINPUTMUX_GpioPort0Pin26ToPintsel

enumerator kINPUTMUX_GpioPort0Pin27ToPintsel

enumerator kINPUTMUX_GpioPort0Pin28ToPintsel

enumerator kINPUTMUX_GpioPort0Pin29ToPintsel

enumerator kINPUTMUX_GpioPort0Pin30ToPintsel

enumerator kINPUTMUX_GpioPort0Pin31ToPintsel

enumerator kINPUTMUX_GpioPort1Pin0ToPintsel

enumerator kINPUTMUX_GpioPort1Pin1ToPintsel

enumerator kINPUTMUX_GpioPort1Pin2ToPintsel

enumerator kINPUTMUX_GpioPort1Pin3ToPintsel

enumerator kINPUTMUX_GpioPort1Pin4ToPintsel

enumerator kINPUTMUX_GpioPort1Pin5ToPintsel

enumerator kINPUTMUX_GpioPort1Pin6ToPintsel

enumerator kINPUTMUX_GpioPort1Pin7ToPintsel

enumerator kINPUTMUX_GpioPort1Pin8ToPintsel

enumerator kINPUTMUX_GpioPort1Pin9ToPintsel

enumerator kINPUTMUX_GpioPort1Pin10ToPintsel

enumerator kINPUTMUX_GpioPort1Pin11ToPintsel

enumerator kINPUTMUX_GpioPort1Pin12ToPintsel

enumerator kINPUTMUX_GpioPort1Pin13ToPintsel

enumerator kINPUTMUX_GpioPort1Pin14ToPintsel

enumerator kINPUTMUX_GpioPort1Pin15ToPintsel

enumerator kINPUTMUX_GpioPort1Pin16ToPintsel

enumerator kINPUTMUX_GpioPort1Pin17ToPintsel

enumerator kINPUTMUX_GpioPort1Pin18ToPintsel

enumerator kINPUTMUX_GpioPort1Pin19ToPintsel

enumerator kINPUTMUX_GpioPort1Pin20ToPintsel

enumerator kINPUTMUX_GpioPort1Pin21ToPintsel

enumerator kINPUTMUX_GpioPort1Pin22ToPintsel

enumerator kINPUTMUX_GpioPort1Pin23ToPintsel

enumerator kINPUTMUX_GpioPort1Pin24ToPintsel

enumerator kINPUTMUX_GpioPort1Pin25ToPintsel

enumerator kINPUTMUX_GpioPort1Pin26ToPintsel

enumerator kINPUTMUX_GpioPort1Pin27ToPintsel

enumerator kINPUTMUX_GpioPort1Pin28ToPintsel

enumerator kINPUTMUX_GpioPort1Pin29ToPintsel

enumerator kINPUTMUX_GpioPort1Pin30ToPintsel

enumerator kINPUTMUX_GpioPort1Pin31ToPintsel
DSP Interrupt.

enumerator kINPUTMUX_Flexcomm0ToDspInterrupt

enumerator kINPUTMUX_Flexcomm1ToDspInterrupt

enumerator kINPUTMUX_Flexcomm2ToDspInterrupt

enumerator kINPUTMUX_Flexcomm3ToDspInterrupt

enumerator kINPUTMUX_Flexcomm4ToDspInterrupt

enumerator kINPUTMUX_Flexcomm5ToDspInterrupt

enumerator kINPUTMUX_Flexcomm6ToDspInterrupt

enumerator kINPUTMUX_Flexcomm7ToDspInterrupt

enumerator kINPUTMUX_GpioInt0ToDspInterrupt

enumerator kINPUTMUX_GpioInt1ToDspInterrupt

enumerator kINPUTMUX_GpioInt2ToDspInterrupt

enumerator kINPUTMUX_GpioInt3ToDspInterrupt

enumerator kINPUTMUX_GpioInt4ToDspInterrupt

enumerator kINPUTMUX_GpioInt5ToDspInterrupt

enumerator kINPUTMUX_GpioInt6ToDspInterrupt

enumerator kINPUTMUX_GpioInt7ToDspInterrupt

enumerator kINPUTMUX_NsHsGpioInt0ToDspInterrupt

enumerator kINPUTMUX_NsHsGpioInt1ToDspInterrupt

enumerator kINPUTMUX_Wdt1ToDspInterrupt

enumerator kINPUTMUX_Dmac0ToDspInterrupt

enumerator kINPUTMUX_Dmac1ToDspInterrupt

enumerator kINPUTMUX_MuBToDspInterrupt

enumerator kINPUTMUX_Utick0ToDspInterrupt

enumerator kINPUTMUX_Mrt0ToDspInterrupt

enumerator kINPUTMUX_OsEventTimerToDspInterrupt

enumerator kINPUTMUX_Ctimer0ToDspInterrupt

enumerator kINPUTMUX_Ctimer1ToDspInterrupt

enumerator kINPUTMUX_Ctimer2ToDspInterrupt

enumerator kINPUTMUX_Ctimer3ToDspInterrupt

enumerator kINPUTMUX_Ctimer4ToDspInterrupt

enumerator kINPUTMUX_RtcToDspInterrupt

enumerator kINPUTMUX_I3c0ToDspInterrupt

enumerator kINPUTMUX_Dmic0ToDspInterrupt

enumerator kINPUTMUX_Hwvad0ToDspInterrupt

enumerator kINPUTMUX_FlexspiToDspInterrupt
Frequency measure.

enumerator kINPUTMUX_XtalinToFreqmeas

enumerator kINPUTMUX_SfroToFreqmeas

enumerator kINPUTMUX_FfroToFreqmeas

enumerator kINPUTMUX_LposcToFreqmeas

enumerator kINPUTMUX_Rtc32KhzOscToFreqmeas

enumerator kINPUTMUX_MainSysClkToFreqmeas

enumerator kINPUTMUX_FreqmeGpioClkToFreqmeas
CTmier0 capture input mux.

enumerator kINPUTMUX_CtInp0ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp1ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp2ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp3ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp4ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp5ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp6ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp7ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp8ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp9ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp10ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp11ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp12ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp13ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp14ToTimer0CaptureChannels

enumerator kINPUTMUX_CtInp15ToTimer0CaptureChannels

enumerator kINPUTMUX_SharedI2s0WsToTimer0CaptureChannels

enumerator kINPUTMUX_SharedI2s1WsToTimer0CaptureChannels

enumerator kINPUTMUX_Usb1FrameToggleToTimer0CaptureChannels
CTmier1 capture input mux.

enumerator kINPUTMUX_CtInp0ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp1ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp2ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp3ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp4ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp5ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp6ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp7ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp8ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp9ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp10ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp11ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp12ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp13ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp14ToTimer1CaptureChannels

enumerator kINPUTMUX_CtInp15ToTimer1CaptureChannels

enumerator kINPUTMUX_SharedI2s0WsToTimer1CaptureChannels

enumerator kINPUTMUX_SharedI2s1WsToTimer1CaptureChannels

enumerator kINPUTMUX_Usb1FrameToggleToTimer1CaptureChannels
CTmier2 capture input mux.

enumerator kINPUTMUX_CtInp0ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp1ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp2ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp3ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp4ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp5ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp6ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp7ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp8ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp9ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp10ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp11ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp12ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp13ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp14ToTimer2CaptureChannels

enumerator kINPUTMUX_CtInp15ToTimer2CaptureChannels

enumerator kINPUTMUX_SharedI2s0WsToTimer2CaptureChannels

enumerator kINPUTMUX_SharedI2s1WsToTimer2CaptureChannels

enumerator kINPUTMUX_Usb1FrameToggleToTimer2CaptureChannels
CTmier3 capture input mux.

enumerator kINPUTMUX_CtInp0ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp1ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp2ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp3ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp4ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp5ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp6ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp7ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp8ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp9ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp10ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp11ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp12ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp13ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp14ToTimer3CaptureChannels

enumerator kINPUTMUX_CtInp15ToTimer3CaptureChannels

enumerator kINPUTMUX_SharedI2s0WsToTimer3CaptureChannels

enumerator kINPUTMUX_SharedI2s1WsToTimer3CaptureChannels

enumerator kINPUTMUX_Usb1FrameToggleToTimer3CaptureChannels

CTmier4 capture input mux.

enumerator kINPUTMUX_CtInp0ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp1ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp2ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp3ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp4ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp5ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp6ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp7ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp8ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp9ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp10ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp11ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp12ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp13ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp14ToTimer4CaptureChannels

enumerator kINPUTMUX_CtInp15ToTimer4CaptureChannels

enumerator kINPUTMUX_SharedI2s0WsToTimer4CaptureChannels

enumerator kINPUTMUX_SharedI2s1WsToTimer4CaptureChannels

enumerator kINPUTMUX_Usb1FrameToggleToTimer4CaptureChannels
DMA0 ITRIG.

enumerator kINPUTMUX_NsGpioPint0Int0ToDma0

enumerator kINPUTMUX_NsGpioPint0Int1ToDma0

enumerator kINPUTMUX_NsGpioPint0Int2ToDma0

enumerator kINPUTMUX_NsGpioPint0Int3ToDma0

enumerator kINPUTMUX_Ctimer0M0ToDma0

enumerator kINPUTMUX_Ctimer0M1ToDma0

enumerator kINPUTMUX_Ctimer1M0ToDma0

enumerator kINPUTMUX_Ctimer1M1ToDma0

enumerator kINPUTMUX_Ctimer2M0ToDma0

enumerator kINPUTMUX_Ctimer2M1ToDma0

enumerator kINPUTMUX_Ctimer3M0ToDma0

enumerator kINPUTMUX_Ctimer3M1ToDma0

enumerator kINPUTMUX_Ctimer4M0ToDma0

enumerator kINPUTMUX_Ctimer4M1ToDma0

enumerator kINPUTMUX_Dma0TrigOutAToDma0

enumerator kINPUTMUX_Dma0TrigOutBToDma0

enumerator kINPUTMUX_Dma0TrigOutCToDma0

enumerator kINPUTMUX_Dma0TrigOutDToDma0

enumerator kINPUTMUX_Sct0Dmac0ToDma0

enumerator kINPUTMUX_Sct0Dmac1ToDma0

enumerator kINPUTMUX_HashCryptOutToDma0

enumerator kINPUTMUX_AcmpToDma0

enumerator kINPUTMUX_AdcToDma0

enumerator kINPUTMUX_FlexspiRxToDma0

enumerator kINPUTMUX_FlexspiTxToDma0
DMA1 ITRIG.

enumerator kINPUTMUX_NsGpioPint0Int0ToDma1

enumerator kINPUTMUX_NsGpioPint0Int1ToDma1

enumerator kINPUTMUX_NsGpioPint0Int2ToDma1

enumerator kINPUTMUX_NsGpioPint0Int3ToDma1

enumerator kINPUTMUX_Ctimer0M0ToDma1

enumerator kINPUTMUX_Ctimer0M1ToDma1

enumerator kINPUTMUX_Ctimer1M0ToDma1

enumerator kINPUTMUX_Ctimer1M1ToDma1

enumerator kINPUTMUX_Ctimer2M0ToDma1

enumerator kINPUTMUX_Ctimer2M1ToDma1

enumerator kINPUTMUX_Ctimer3M0ToDma1

enumerator kINPUTMUX_Ctimer3M1ToDma1

enumerator kINPUTMUX_Ctimer4M0ToDma1

enumerator kINPUTMUX_Ctimer4M1ToDma1

enumerator kINPUTMUX_Dma1TrigOutAToDma1

enumerator kINPUTMUX_Dma1TrigOutBToDma1

enumerator kINPUTMUX_Dma1TrigOutCToDma1

enumerator kINPUTMUX_Dma1TrigOutDToDma1

enumerator kINPUTMUX_Sct0Dmac0ToDma1

enumerator kINPUTMUX_Sct0Dmac1ToDma1

enumerator kINPUTMUX_HashCryptOutToDma1

enumerator kINPUTMUX_AcmpToDma1

enumerator kINPUTMUX_AdcToDma1

enumerator kINPUTMUX_FlexspiRxToDma1

enumerator kINPUTMUX_FlexspiTxToDma1
DMA0 OTRIG.

enumerator kINPUTMUX_Dma0OtrigChannel0ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel1ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel2ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel3ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel4ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel5ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel6ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel7ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel8ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel9ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel10ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel11ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel12ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel13ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel14ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel15ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel16ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel17ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel18ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel19ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel20ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel21ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel22ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel23ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel24ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel25ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel26ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel27ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel28ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel29ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel30ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel31ToTriginChannels

enumerator kINPUTMUX_Dma0OtrigChannel32ToTriginChannels
DMA1 OTRIG.

enumerator kINPUTMUX_Dma1OtrigChannel0ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel1ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel2ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel3ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel4ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel5ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel6ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel7ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel8ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel9ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel10ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel11ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel12ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel13ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel14ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel15ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel16ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel17ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel18ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel19ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel20ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel21ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel22ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel23ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel24ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel25ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel26ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel27ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel28ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel29ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel30ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel31ToTriginChannels

enumerator kINPUTMUX_Dma1OtrigChannel32ToTriginChannels

enum _inputmux_signal_t

INPUTMUX signal enable/disable type.

Values:

enumerator kINPUTMUX_Dmac0InputTriggerPint0Ena

DMA0 input trigger source enable.

enumerator kINPUTMUX_Dmac0InputTriggerPint1Ena

enumerator kINPUTMUX_Dmac0InputTriggerPint2Ena

enumerator kINPUTMUX_Dmac0InputTriggerPint3Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer0M0Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer0M1Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer1M0Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer1M1Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer2M0Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer2M1Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer3M0Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer3M1Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer4M0Ena

enumerator kINPUTMUX_Dmac0InputTriggerCtimer4M1Ena

enumerator kINPUTMUX_Dmac0InputTriggerDma0OutAEna

enumerator kINPUTMUX_Dmac0InputTriggerDma0OutBEna

enumerator kINPUTMUX_Dmac0InputTriggerDma0OutCEna

enumerator kINPUTMUX_Dmac0InputTriggerDma0OutDEna

enumerator kINPUTMUX_Dmac0InputTriggerSctDmac0Ena

enumerator kINPUTMUX_Dmac0InputTriggerSctDmac1Ena

enumerator kINPUTMUX_Dmac0InputTriggerHashOutEna

enumerator kINPUTMUX_Dmac0InputTriggerAcmpEna

enumerator kINPUTMUX_Dmac0InputTriggerAdcEna

enumerator kINPUTMUX_Dmac0InputTriggerFlexspiRxEna

enumerator kINPUTMUX_Dmac0InputTriggerFlexspiTxEna
DMA1 input trigger source enable.

enumerator kINPUTMUX_Dmac1InputTriggerPint0Ena

enumerator kINPUTMUX_Dmac1InputTriggerPint1Ena

enumerator kINPUTMUX_Dmac1InputTriggerPint2Ena

enumerator kINPUTMUX_Dmac1InputTriggerPint3Ena

enumerator kINPUTMUX_Dmac1InputTriggerCtimer0M0Ena

enumerator kINPUTMUX_Dmac1InputTriggerCtimer0M1Ena

enumerator kINPUTMUX_Dmac1InputTriggerCtimer1M0Ena

enumerator kINPUTMUX_Dmac1InputTriggerCtimer1M1Ena

enumerator kINPUTMUX_Dmac1InputTriggerCtimer2M0Ena

enumerator kINPUTMUX_Dmac1InputTriggerCtimer2M1Ena

enumerator kINPUTMUX_Dmac1InputTriggerCtimer3M0Ena

enumerator kINPUTMUX_Dmac1InputTriggerCtimer3M1Ena

enumerator kINPUTMUX_Dmac1InputTriggerCtimer4M0Ena

enumerator kINPUTMUX_Dmac1InputTriggerCtimer4M1Ena

enumerator kINPUTMUX_Dmac1InputTriggerDma1OutAEna

enumerator kINPUTMUX_Dmac1InputTriggerDma1OutBEna

enumerator kINPUTMUX_Dmac1InputTriggerDma1OutCEna

enumerator kINPUTMUX_Dmac1InputTriggerDma1OutDEna

enumerator kINPUTMUX_Dmac1InputTriggerSctDmac0Ena

enumerator kINPUTMUX_Dmac1InputTriggerSctDmac1Ena

enumerator kINPUTMUX_Dmac1InputTriggerHashOutEna

enumerator kINPUTMUX_Dmac1InputTriggerAcmpEna

enumerator kINPUTMUX_Dmac1InputTriggerAdcEna

enumerator kINPUTMUX_Dmac1InputTriggerFlexspiRxEna

enumerator kINPUTMUX_Dmac1InputTriggerFlexspiTxEna
DMA0 REQ signal.

enumerator kINPUTMUX_Flexcomm0RxToDmac0Ch0RequestEna

enumerator kINPUTMUX_Flexcomm0TxToDmac0Ch1RequestEna

enumerator kINPUTMUX_Flexcomm1RxToDmac0Ch2RequestEna

enumerator kINPUTMUX_Flexcomm1TxToDmac0Ch3RequestEna

enumerator kINPUTMUX_Flexcomm2RxToDmac0Ch4RequestEna

enumerator kINPUTMUX_Flexcomm2TxToDmac0Ch5RequestEna

enumerator kINPUTMUX_Flexcomm3RxToDmac0Ch6RequestEna

enumerator kINPUTMUX_Flexcomm3TxToDmac0Ch7RequestEna

enumerator kINPUTMUX_Flexcomm4RxToDmac0Ch8RequestEna

enumerator kINPUTMUX_Flexcomm4TxToDmac0Ch9RequestEna

enumerator kINPUTMUX_Flexcomm5RxToDmac0Ch10RequestEna

enumerator kINPUTMUX_Flexcomm5TxToDmac0Ch11RequestEna

enumerator kINPUTMUX_Flexcomm6RxToDmac0Ch12RequestEna

enumerator kINPUTMUX_Flexcomm6TxToDmac0Ch13RequestEna

enumerator kINPUTMUX_Flexcomm7RxToDmac0Ch14RequestEna

enumerator kINPUTMUX_Flexcomm7TxToDmac0Ch15RequestEna

enumerator kINPUTMUX_Dmic0Ch0ToDmac0Ch16RequestEna

enumerator kINPUTMUX_Dmic0Ch1ToDmac0Ch17RequestEna

enumerator kINPUTMUX_Dmic0Ch2ToDmac0Ch18RequestEna

enumerator kINPUTMUX_Dmic0Ch3ToDmac0Ch19RequestEna

enumerator kINPUTMUX_Dmic0Ch4ToDmac0Ch20RequestEna

enumerator kINPUTMUX_Dmic0Ch5ToDmac0Ch21RequestEna

enumerator kINPUTMUX_Dmic0Ch6ToDmac0Ch22RequestEna

enumerator kINPUTMUX_Dmic0Ch7ToDmac0Ch23RequestEna

enumerator kINPUTMUX_I3c0RxToDmac0Ch24RequestEna

enumerator kINPUTMUX_I3c0TxToDmac0Ch25RequestEna

enumerator kINPUTMUX_Flexcomm14RxToDmac0Ch26RequestEna

enumerator kINPUTMUX_Flexcomm14TxToDmac0Ch27RequestEna

enumerator kINPUTMUX_HashCryptToDmac0Ch30RequestEna
DMA1 REQ signal.

enumerator kINPUTMUX_Flexcomm0RxToDmac1Ch0RequestEna

enumerator kINPUTMUX_Flexcomm0TxToDmac1Ch1RequestEna

enumerator kINPUTMUX_Flexcomm1RxToDmac1Ch2RequestEna

enumerator kINPUTMUX_Flexcomm1TxToDmac1Ch3RequestEna

enumerator kINPUTMUX_Flexcomm2RxToDmac1Ch4RequestEna

enumerator kINPUTMUX_Flexcomm2TxToDmac1Ch5RequestEna

enumerator kINPUTMUX_Flexcomm3RxToDmac1Ch6RequestEna

enumerator kINPUTMUX_Flexcomm3TxToDmac1Ch7RequestEna

enumerator kINPUTMUX_Flexcomm4RxToDmac1Ch8RequestEna

enumerator kINPUTMUX_Flexcomm4TxToDmac1Ch9RequestEna

enumerator kINPUTMUX_Flexcomm5RxToDmac1Ch10RequestEna

enumerator kINPUTMUX_Flexcomm5TxToDmac1Ch11RequestEna

enumerator kINPUTMUX_Flexcomm6RxToDmac1Ch12RequestEna

enumerator kINPUTMUX_Flexcomm6TxToDmac1Ch13RequestEna

enumerator kINPUTMUX_Flexcomm7RxToDmac1Ch14RequestEna

enumerator kINPUTMUX_Flexcomm7TxToDmac1Ch15RequestEna

enumerator kINPUTMUX_Dmic0Ch0ToDmac1Ch16RequestEna

enumerator kINPUTMUX_Dmic0Ch1ToDmac1Ch17RequestEna

enumerator kINPUTMUX_Dmic0Ch2ToDmac1Ch18RequestEna

enumerator kINPUTMUX_Dmic0Ch3ToDmac1Ch19RequestEna

enumerator kINPUTMUX_Dmic0Ch4ToDmac1Ch20RequestEna

enumerator kINPUTMUX_Dmic0Ch5ToDmac1Ch21RequestEna

enumerator kINPUTMUX_Dmic0Ch6ToDmac1Ch22RequestEna

enumerator kINPUTMUX_Dmic0Ch7ToDmac1Ch23RequestEna

enumerator kINPUTMUX_I3c0RxToDmac1Ch24RequestEna

enumerator kINPUTMUX_I3c0TxToDmac1Ch25RequestEna

enumerator kINPUTMUX_Flexcomm14RxToDmac1Ch26RequestEna

enumerator kINPUTMUX_Flexcomm14TxToDmac1Ch27RequestEna

enumerator kINPUTMUX_HashCryptToDmac1Ch30RequestEna

typedef enum *_inputmux_connection_t* inputmux_connection_t

INPUTMUX connections type.

typedef enum *_inputmux_signal_t* inputmux_signal_t

INPUTMUX signal enable/disable type.

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_EnableSignal(void *base, *inputmux_signal_t* signal, bool enable)

Enable/disable a signal.

This function gates the INPUTMUX clock.

Parameters

- base – Base address of the INPUTMUX peripheral.
- signal – Enable signal register id and bit offset.
- enable – Selects enable or disable.

void INPUTMUX_Deinit(void *base)

Deinitialize INPUTMUX peripheral.

This function disables the INPUTMUX clock.

Parameters

- base – Base address of the INPUTMUX peripheral.

SCT0_PMUX_ID

Periphinmux IDs.

SHSGPIO_PMUX_ID

PINTSEL_PMUX_ID

DSP_INT_PMUX_ID

DMA0_ITRIG_PMUX_ID

DMA0_OTRIG_PMUX_ID

DMA1_ITRIG_PMUX_ID

DMA1_OTRIG_PMUX_ID

CT32BIT0_CAP_PMUX_ID

CT32BIT1_CAP_PMUX_ID

CT32BIT2_CAP_PMUX_ID

CT32BIT3_CAP_PMUX_ID

CT32BIT4_CAP_PMUX_ID

FREQMEAS_PMUX_ID

DMA0_REQ_ENA0_ID

DMA1_REQ_ENA0_ID

DMA0_ITRIG_EN0_ID

DMA1_ITRIG_EN0_ID

ENA_SHIFT

PMUX_SHIFT

2.44 IOPCTL: Input/Output Pad Controller

LPC_IOPCTL_DRIVER_VERSION
: IOPCTL driver version 2.0.0.

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 (IOPCTL_Type *base, uint8_t port, uint8_t pin, uint32_t modefunc)

Sets I/O Pad Control pin mux.

Parameters

- base – : The base of IOPCTL peripheral on the chip
- port – : Port to mux
- pin – : Pin to mux
- modefunc – : OR'ed values of type IOPCTL_*

Returns

Nothing

___STATIC_INLINE void IOPCTL_SetPinMuxing (IOPCTL_Type *base, const iopctl_group_t *pinArray, uint32_t arrayLength)

Set all I/O Control pin muxing.

Parameters

- base – : The base of IOPCTL peripheral on the chip
- pinArray – : Pointer to array of pin mux selections
- arrayLength – : Number of entries in pinArray

Returns

Nothing

FSL_COMPONENT_ID

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_FULLDRIVE_EN
: Selects full drive

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

struct _iopctl_group
: *#include <fsl_iopctl.h>* Array of IOPCTL pin definitions passed to *IOPCTL_SetPinMuxing()* must be in this format.

2.45 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

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 define a variable with L1 d-cache line size alignment

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_SIZEALIGN(var, alignbytes)

Macro to define a variable with L1 d-cache line size alignment

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

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*.

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*.

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.

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.

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.

Empty function for build compatibility.

Returns

Always return 0;

static inline void EnableGlobalIRQ(uint32_t primask)

Enable the global IRQ.

Empty function for build compatibility.

Parameters

- primask – Not used.

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.

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.

2.46 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 GPIO: General Purpose I/O

void GPIO_PortInit(GPIO_Type *base, uint32_t port)

Initializes the GPIO peripheral.

This function ungates the GPIO clock.

Parameters

- base – GPIO peripheral base pointer.
- port – GPIO port number.

void GPIO_PinInit(GPIO_Type *base, uint32_t port, 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, either input or output, in the user file. Then, call the *GPIO_PinInit()* function.

This is an example to define an input pin or 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(Typically GPIO)
- port – GPIO port number
- pin – GPIO pin number
- config – GPIO pin configuration pointer

static inline void GPIO_PinWrite(GPIO_Type *base, uint32_t port, uint32_t pin, uint8_t output)

Sets the output level of the one GPIO pin to the logic 1 or 0.

Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- 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 uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t port, uint32_t pin)

Reads the current input value of the GPIO PIN.

Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- pin – GPIO pin number

Return values

GPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

FSL_GPIO_DRIVER_VERSION
LPC GPIO driver version.

enum _gpio_pin_direction
LPC GPIO direction definition.
Values:

enumerator kGPIO_DigitalInput
Set current pin as digital input

enumerator kGPIO_DigitalOutput
Set current pin as digital output

enum _gpio_pin_enable_mode
GPIO Pin Interrupt enable mode.
Values:

enumerator kGPIO_PinIntEnableLevel
Generate Pin Interrupt on level mode

enumerator kGPIO_PinIntEnableEdge
Generate Pin Interrupt on edge mode

enum _gpio_pin_enable_polarity
GPIO Pin Interrupt enable polarity.
Values:

enumerator kGPIO_PinIntEnableHighOrRise
Generate Pin Interrupt on high level or rising edge

enumerator kGPIO_PinIntEnableLowOrFall
Generate Pin Interrupt on low level or falling edge

enum _gpio_interrupt_index
LPC GPIO interrupt index definition.
Values:

enumerator kGPIO_InterruptA
Set current pin as interrupt A

enumerator kGPIO_InterruptB
Set current pin as interrupt B

typedef enum *_gpio_pin_direction* gpio_pin_direction_t
LPC GPIO direction definition.

typedef struct *_gpio_pin_config* gpio_pin_config_t

The GPIO pin configuration structure.

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

typedef enum *_gpio_pin_enable_mode* gpio_pin_enable_mode_t

GPIO Pin Interrupt enable mode.

typedef enum *_gpio_pin_enable_polarity* gpio_pin_enable_polarity_t

GPIO Pin Interrupt enable polarity.

typedef enum *_gpio_interrupt_index* gpio_interrupt_index_t

LPC GPIO interrupt index definition.

typedef struct *_gpio_interrupt_config* gpio_interrupt_config_t

Configures the interrupt generation condition.

static inline void GPIO_PortSet(GPIO_Type *base, uint32_t port, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- mask – GPIO pin number macro

static inline void GPIO_PortClear(GPIO_Type *base, uint32_t port, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- mask – GPIO pin number macro

static inline void GPIO_PortToggle(GPIO_Type *base, uint32_t port, uint32_t mask)

Reverses current output logic of the multiple GPIO pins.

Parameters

- base – GPIO peripheral base pointer(Typically GPIO)
- port – GPIO port number
- mask – GPIO pin number macro

GPIO_PIN_INT_LEVEL

GPIO_PIN_INT_EDGE

PINT_PIN_INT_HIGH_OR_RISE_TRIGGER

PINT_PIN_INT_LOW_OR_FALL_TRIGGER

struct _gpio_pin_config

#include <fsl_gpio.h> The GPIO pin configuration structure.

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

Public Members

gpio_pin_direction_t pinDirection

GPIO direction, input or output

uint8_t outputLogic

Set default output logic, no use in input

struct _gpio_interrupt_config

#include <fsl_gpio.h> Configures the interrupt generation condition.

2.48 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

2.49 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 MU_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 MU_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)

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 MU_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 – MU peripheral base address.
- flags – The 3-bit MU flags to set.

Return values

- kStatus_Success – Flags were set successfully.
- kStatus_Timeout – Timeout occurred while waiting for flags to update.

Returns
status_t

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.

static inline 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);
}
```

Parameters

- base – MU peripheral base address.

Returns
Bit mask of the MU status flags, see _mu_status_flags.

static inline uint32_t MU_GetRxStatusFlags(MU_Type *base)

Return the RX status flags.

This function return the RX status flags. Note: RFn bits of SR[27-24](mu status register) are mapped in reverse numerical order: RF0 -> SR[27] RF1 -> SR[26] RF2 -> SR[25] RF3 -> SR[24]

```
status_reg = MU_GetRxStatusFlags(base);
```

Parameters

- base – MU peripheral base address.

Returns
MU RX status

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 mask)

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);
```

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU status flags. See _mu_status_flags. The following flags are cleared by hardware, this function could not clear them.
  - kMU_Tx0EmptyFlag
  - kMU_Tx1EmptyFlag
  - kMU_Tx2EmptyFlag
  - kMU_Tx3EmptyFlag
  - kMU_Rx0FullFlag
  - kMU_Rx1FullFlag
  - kMU_Rx2FullFlag
  - kMU_Rx3FullFlag
  - kMU_EventPendingFlag
  - kMU_FlagsUpdatingFlag
  - kMU_OtherSideInResetFlag

static inline void MU_EnableInterrupts(MU_Type *base, uint32_t mask)

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);
```

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU interrupts. See _mu_interrupt_enable.

static inline void MU_DisableInterrupts(MU_Type *base, uint32_t mask)

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);
```

Parameters

- base – MU peripheral base address.
- mask – Bit mask of the MU interrupts. See _mu_interrupt_enable.

status_t MU_TriggerInterrupts(MU_Type *base, uint32_t mask)

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.
}
```

Parameters

- base – MU peripheral base address.
- mask – 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_MaskHardwareReset(MU_Type *base, bool mask)

Mask hardware reset by the other core.

The other core could call MU_HardwareResetOtherCore() to reset current core. To mask the reset, call this function and pass in true.

Parameters

- base – MU peripheral base address.
- mask – Pass true to mask the hardware reset, pass false to unmask it.

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_EventPendingFlag
MU event pending.

enumerator kMU_FlagsUpdatingFlag
MU flags update is on-going.

enumerator kMU_ResetAssertInterruptFlag
The other core reset assert interrupt pending.

enumerator kMU_ResetDeassertInterruptFlag
The other core reset de-assert interrupt pending.

enumerator kMU_OtherSideInResetFlag
The other side is in reset.

enumerator kMU_MuResetInterruptFlag
The other side initializes MU reset.

enumerator kMU_HardwareResetInterruptFlag
Current side has been hardware reset by the other side.

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_ResetAssertInterruptEnable

The other core reset assert interrupt.

enumerator kMU_ResetDeassertInterruptEnable

The other core reset de-assert interrupt.

enumerator kMU_MuResetInterruptEnable

The other side initializes MU reset. The interrupt is ORed with the general purpose interrupt 3. The general purpose interrupt 3 is issued when the other side set the MU reset and this interrupt is enabled.

enumerator kMU_HardwareResetInterruptEnable

Current side has been hardware reset by the other side.

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.

Values:

enumerator kMU_MsgReg0

enumerator kMU_MsgReg1

enumerator kMU_MsgReg2

enumerator kMU_MsgReg3

typedef enum *_mu_msg_reg_index* mu_msg_reg_index_t

MU message register.

MU_CR_NMI_MASK

MU_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.

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)

2.50 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.

2.51 OTFAD: On The Fly AES-128 Decryption Driver

void OTFAD_GetDefaultConfig(*otfad_config_t* *config)
OTFAD module initialization function.

Parameters

- config – OTFAD configuration.

void OTFAD_Init(OTFAD_Type *base, const *otfad_config_t* *config)
OTFAD module initialization function.

Parameters

- base – OTFAD base address.
- config – OTFAD configuration.

void OTFAD_Deinit(OTFAD_Type *base)
Deinitializes the OTFAD.

static inline uint32_t OTFAD_GetOperateMode(OTFAD_Type *base)
OTFAD module get operate mode.

Parameters

- base – OTFAD base address.

static inline uint32_t OTFAD_GetStatus(OTFAD_Type *base)
OTFAD module get status.

Parameters

- base – OTFAD base address.

status_t OTFAD_SetEncryptionConfig(OTFAD_Type *base, const *otfad_encryption_config_t* *config)
OTFAD module set encryption configuration.

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

Parameters

- base – OTFAD base address.

- config – encryption configuration.

status_t OTFAD_GetEncryptionConfig(OTFAD_Type *base, *otfad_encryption_config_t* *config)

OTFAD module get encryption configuration.

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

Parameters

- base – OTFAD base address.
- config – encryption configuration.

status_t OTFAD_HitDetermination(OTFAD_Type *base, uint32_t address, uint8_t *contextIndex)

OTFAD module do hit determination.

Parameters

- base – OTFAD base address.
- address – the physical address space assigned to the QuadSPI(FlexSPI) module.
- contextIndex – hitted context region index.

Returns

status, such as kStatus_Success or kStatus_OTFAD_ResRegAccessMode.

FSL_OTFAD_DRIVER_VERSION

Driver version.

Status codes for the OTFAD driver.

Values:

enumerator kStatus_OTFAD_ResRegAccessMode

Restricted register mode

enumerator kStatus_OTFAD_AddressError

End address less than start address

enumerator kStatus_OTFAD_RegionOverlap

the OTFAD does not support any form of memory region overlap, for system accesses that hit in multiple contexts or no contexts, the fetched data is simply bypassed

enumerator kStatus_OTFAD_RegionMiss

For accesses that hit in a single context, but not the selected one

OTFAD context type.

Values:

enumerator kOTFAD_Context_0

context 0

enumerator kOTFAD_Context_1
context 1

enumerator kOTFAD_Context_2
context 2

enumerator kOTFAD_Context_3
context 3

OTFAD operate mode.
Values:

enumerator kOTFAD_NRM
Normal Mode

enumerator kOTFAD_SVM
Security Violation Mode

enumerator kOTFAD_LDM
Logically Disabled Mode

typedef struct *_otfad_encryption_config* otfad_encryption_config_t
OTFAD encryption configuration structure.

typedef struct *_otfad_config* otfad_config_t
OTFAD configuration structure.

struct _otfad_encryption_config
#include <fsl_otfad.h> OTFAD encryption configuration structure.

Public Members

bool valid
The context is valid or not

bool AESdecryption
AES decryption enable

uint8_t readOnly
read write attribute for the entire set of context registers

uint8_t contextIndex
OTFAD context index

uint32_t startAddr
Start address

uint32_t endAddr
End address

uint32_t key[4]
Encryption key

uint32_t counter[2]
Encryption counter

struct _otfad_config
#include <fsl_otfad.h> OTFAD configuration structure.

Public Members

bool enableIntRequest
Interrupt request enable

bool forceError
Forces the OTFAD's key blob error flag (SR[KBERR]) to be asserted

bool forceSVM
Force entry into SVM after a write

bool forceLDM
Force entry into LDM after a write

bool keyBlobScramble
Key blob KEK scrambling

bool keyBlobProcess
Key blob processing

bool startKeyBlobProcessing
key blob processing is initiated

bool restrictedRegAccess
Restricted register access enable

bool enableOTFAD
OTFAD has decryption enabled

2.52 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>

2.53 Power Driver

enum _pmc_interrupt
PMC event flags.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kPMC_INT_LVDCORE
Vddcore Low-Voltage Detector Interrupt Enable.

enumerator kPMC_INT_HVDCORE
Vddcore High-Voltage Detector Interrupt Enable.

enumerator kPMC_INT_HVD1V8
Vdd1v8 High-Voltage Detector Interrupt Enable.

enumerator kPMC_INT_AUTOWK
PMC automatic wakeup enable and interrupt enable.

enumerator kPMC_INT_INTRPAD
Interrupt pad deep powerdown and deep sleep wake up & interrupt enable.

enum _pmc_event_flags
PMC event flags.

Note: These enums are meant to be OR'd together to form a bit mask.

Values:

enumerator kPMC_FLAGS_PORCORE
POR triggered by the vddcore POR monitor (0 = no, 1 = yes).

enumerator kPMC_FLAGS_POR1V8
vdd1v8 power on event detected since last cleared(0 = no, 1 = yes).

enumerator kPMC_FLAGS_PORAO18
vdd_ao18 power on event detected since last cleared (0 = no, 1 = yes).

enumerator kPMC_FLAGS_LVDCORE
LVD tripped since last time this bit was cleared (0 = no, 1 = yes).

enumerator kPMC_FLAGS_HVDCORE
HVD tripped since last time this bit was cleared (0 = no, 1 = yes).

enumerator kPMC_FLAGS_HVD1V8
vdd1v8 HVD tripped since last time this bit was cleared (0 = no, 1 = yes).

enumerator kPMC_FLAGS_RTC
RTC wakeup detected since last time flag was cleared (0 = no, 1 = yes).

enumerator kPMC_FLAGS_AUTOWK
PMC Auto wakeup caused a deep sleep wakeup and interrupt (0 = no, 1 = yes).

enumerator kPMC_FLAGS_INTNPADF
Pad interrupt caused a wakeup or interrupt event since the last time this flag was cleared (0 = no, 1 = yes).

enumerator kPMC_FLAGS_RESETNPAD

Reset pad wakeup caused a wakeup or reset event since the last time this bit was cleared. (0 = no, 1 = yes).

enumerator kPMC_FLAGS_DEEPPD

Deep powerdown was entered since the last time this flag was cleared (0 = no, 1 = yes).

enum pd_bits

Values:

enumerator kPDRUNCFG_PMC_MODE0

enumerator kPDRUNCFG_PMC_MODE1

enumerator kPDRUNCFG_LP_VDD_COREREG

enumerator kPDRUNCFG_LP_PMCREF

enumerator kPDRUNCFG_PD_HVD1V8

enumerator kPDRUNCFG_LP_PORCORE

enumerator kPDRUNCFG_LP_LVDCORE

enumerator kPDRUNCFG_PD_HVDCORE

enumerator kPDRUNCFG_PD_SYSXTAL

enumerator kPDRUNCFG_PD_LPOSC

enumerator kPDRUNCFG_PD_SFRO

enumerator kPDRUNCFG_PD_FFRO

enumerator kPDRUNCFG_PD_SYSPLL_LDO

enumerator kPDRUNCFG_PD_SYSPLL_ANA

enumerator kPDRUNCFG_PD_AUDPLL_LDO

enumerator kPDRUNCFG_PD_AUDPLL_ANA

enumerator kPDRUNCFG_PD_ADC

enumerator kPDRUNCFG_LP_ADC

enumerator kPDRUNCFG_PD_ADC_TEMPSNS

enumerator kPDRUNCFG_PD_ACMP

enumerator kPDRUNCFG_LP_HSPAD_VDET

enumerator kPDRUNCFG_PD_HSPAD_REF

enumerator kPDRUNCFG_APD_PQ_SRAM

enumerator kPDRUNCFG_PPD_PQ_SRAM

enumerator kPDRUNCFG_APD_FLEXSPI_SRAM

enumerator kPDRUNCFG_PPD_FLEXSPI_SRAM

enumerator kPDRUNCFG_APD_USBHS_SRAM

enumerator kPDRUNCFG_PPD_USBHS_SRAM

enumerator kPDRUNCFG_APD_USDHC0_SRAM

enumerator kPDRUNCFG_PPD_USDHC0_SRAM

enumerator kPDRUNCFG_APD_USDHC1_SRAM

enumerator kPDRUNCFG_PPD_USDHC1_SRAM

enumerator kPDRUNCFG_APD_CASPER_SRAM

enumerator kPDRUNCFG_PPD_CASPER_SRAM

enumerator kPDRUNCFG_APD_DSP_CACHE_REGF

enumerator kPDRUNCFG_PPD_DSP_CACHE_REGF

enumerator kPDRUNCFG_APD_DSP_TCM_REGF

enumerator kPDRUNCFG_PPD_DSP_TCM_REGF

enumerator kPDRUNCFG_PD_ROM

enumerator kPDRUNCFG_SRAM_SLEEP

enumerator kPDRUNCFG_APD_SRAM_IF0

enumerator kPDRUNCFG_APD_SRAM_IF1

enumerator kPDRUNCFG_APD_SRAM_IF2

enumerator kPDRUNCFG_APD_SRAM_IF3

enumerator kPDRUNCFG_APD_SRAM_IF4

enumerator kPDRUNCFG_APD_SRAM_IF5

enumerator kPDRUNCFG_APD_SRAM_IF6

enumerator kPDRUNCFG_APD_SRAM_IF7

enumerator kPDRUNCFG_APD_SRAM_IF8

enumerator kPDRUNCFG_APD_SRAM_IF9

enumerator kPDRUNCFG_APD_SRAM_IF10

enumerator kPDRUNCFG_APD_SRAM_IF11

enumerator kPDRUNCFG_APD_SRAM_IF12

enumerator kPDRUNCFG_APD_SRAM_IF13

enumerator kPDRUNCFG_APD_SRAM_IF14

enumerator kPDRUNCFG_APD_SRAM_IF15

enumerator kPDRUNCFG_APD_SRAM_IF16

enumerator kPDRUNCFG_APD_SRAM_IF17

enumerator kPDRUNCFG_APD_SRAM_IF18

enumerator kPDRUNCFG_APD_SRAM_IF19

enumerator kPDRUNCFG_APD_SRAM_IF20

enumerator kPDRUNCFG_APD_SRAM_IF21

enumerator kPDRUNCFG_APD_SRAM_IF22

enumerator kPDRUNCFG_APD_SRAM_IF23

enumerator kPDRUNCFG_APD_SRAM_IF24

enumerator kPDRUNCFG_APD_SRAM_IF25

enumerator kPDRUNCFG_APD_SRAM_IF26

enumerator kPDRUNCFG_APD_SRAM_IF27

enumerator kPDRUNCFG_APD_SRAM_IF28

enumerator kPDRUNCFG_APD_SRAM_IF29

enumerator kPDRUNCFG_PPD_SRAM_IF0

enumerator kPDRUNCFG_PPD_SRAM_IF1

enumerator kPDRUNCFG_PPD_SRAM_IF2

enumerator kPDRUNCFG_PPD_SRAM_IF3

enumerator kPDRUNCFG_PPD_SRAM_IF4

enumerator kPDRUNCFG_PPD_SRAM_IF5

enumerator kPDRUNCFG_PPD_SRAM_IF6

enumerator kPDRUNCFG_PPD_SRAM_IF7

enumerator kPDRUNCFG_PPD_SRAM_IF8

enumerator kPDRUNCFG_PPD_SRAM_IF9

enumerator kPDRUNCFG_PPD_SRAM_IF10

enumerator kPDRUNCFG_PPD_SRAM_IF11

enumerator kPDRUNCFG_PPD_SRAM_IF12

enumerator kPDRUNCFG_PPD_SRAM_IF13

enumerator kPDRUNCFG_PPD_SRAM_IF14

enumerator kPDRUNCFG_PPD_SRAM_IF15

enumerator kPDRUNCFG_PPD_SRAM_IF16

enumerator kPDRUNCFG_PPD_SRAM_IF17

enumerator kPDRUNCFG_PPD_SRAM_IF18

enumerator kPDRUNCFG_PPD_SRAM_IF19

enumerator kPDRUNCFG_PPD_SRAM_IF20

enumerator kPDRUNCFG_PPD_SRAM_IF21

enumerator kPDRUNCFG_PPD_SRAM_IF22

enumerator kPDRUNCFG_PPD_SRAM_IF23

enumerator kPDRUNCFG_PPD_SRAM_IF24

enumerator kPDRUNCFG_PPD_SRAM_IF25

enumerator kPDRUNCFG_PPD_SRAM_IF26

enumerator kPDRUNCFG_PPD_SRAM_IF27

enumerator kPDRUNCFG_PPD_SRAM_IF28

enumerator kPDRUNCFG_PPD_SRAM_IF29

enumerator kPDRUNCFG_ForceUnsigned

enum _power_mode_config

Power mode configuration API parameter.

Values:

enumerator kPmu_Sleep

enumerator kPmu_Deep_Sleep

enumerator kPmu_Deep_PowerDown

enumerator kPmu_Full_Deep_PowerDown

enum _body_bias_mode

Body Bias mode definition.

Values:

enumerator kPmu_Fbb

enumerator kPmu_Rbb

enumerator kPmu_Nbb

enum _pmic_mode_reg

Values:

enumerator kCfg_Run

enumerator kCfg_Sleep

enum _power_pad_vrange_val

pad voltage range value.

Values:

enumerator kPadVol_171_360

Voltage from 1.71V to 3.60V.

enumerator kPadVol_171_198

Voltage from 1.71V to 1.98V.

enumerator kPadVol_300_360

Voltage from 3.00V to 3.60V.

enum _power_lvd_falling_trip_vol_val

LVD falling trip voltage value.

Values:

enumerator kLvdFallingTripVol_720

Voltage 720mV.

enumerator kLvdFallingTripVol_735

Voltage 735mV.

enumerator kLvdFallingTripVol_750

Voltage 750mV.

enumerator kLvdFallingTripVol_765

Voltage 765mV.

enumerator kLvdFallingTripVol_780

Voltage 780mV.

enumerator kLvdFallingTripVol_795

Voltage 795mV.

enumerator kLvdFallingTripVol_810

Voltage 810mV.

enumerator kLvdFallingTripVol_825
Voltage 825mV.

enumerator kLvdFallingTripVol_840
Voltage 840mV.

enumerator kLvdFallingTripVol_855
Voltage 855mV.

enumerator kLvdFallingTripVol_870
Voltage 870mV.

enumerator kLvdFallingTripVol_885
Voltage 885mV.

enumerator kLvdFallingTripVol_900
Voltage 900mV.

enumerator kLvdFallingTripVol_915
Voltage 915mV.

enumerator kLvdFallingTripVol_930
Voltage 930mV.

enumerator kLvdFallingTripVol_945
Voltage 945mV.

enum _power_part_temp_range
Part temperature range.

Values:

enumerator kPartTemp_0C_P85C
Part temp range 0C - 85C.

enumerator kPartTemp_N20C_P85C
Part temp range -20C - 85C.

enum _power_volt_op_range
Voltage operation range.

Values:

enumerator kVoltOpLowRange
Voltage operation range is (0.7V, 0.8V, 0.9V). Maximum supported CM33 frequency is 220MHz for 0C-85C part and 215MHz for -20C-85C part. Maximum supported DSP frequency is 375MHz for 0C-85C part and 355MHz for -20C-85C part.

enumerator kVoltOpFullRange
Voltage operation range is (0.7V, 0.8V, 0.9V, 1.0V, 1.13V). This range can support full CM33/DSP speed clarified in Data Sheet.

enum _power_vddcore_src

VDDCORE supply source.

Values:

enumerator kVddCoreSrc_LDO

VDDCORE is supplied by onchip regulator.

enumerator kVddCoreSrc_PMIC

VDDCORE is supplied by external PMIC.

enum _power_control_for_pmic_mode

vddcore or vdd1v8 power on selection for different PMIC mode. vddcore and vdd1v8 are always on in mode0. Refer to PMC->PMICCFG.

Values:

enumerator kVddCoreOnMode1

VDDCORE is powered in PMIC mode1.

enumerator kVddCoreOnMode2

VDDCORE is powered in PMIC mode2.

enumerator kVddCoreOnMode3

VDDCORE is powered in PMIC mode3.

enumerator kVdd1v8OnMode1

VDD1V8 is powered in PMIC mode1.

enumerator kVdd1v8OnMode2

VDD1V8 is powered in PMIC mode2.

enumerator kVdd1v8OnMode3

VDD1V8 is powered in PMIC mode3.

typedef enum *pd_bits* pd_bit_t

typedef enum *_power_mode_config* power_mode_cfg_t

Power mode configuration API parameter.

typedef enum *_body_bias_mode* body_bias_mode_t

Body Bias mode definition.

typedef enum *_pmic_mode_reg* pmic_mode_reg_t

typedef enum *_power_pad_vrange_val* power_pad_vrange_val_t

pad voltage range value.

typedef struct *_power_pad_vrange* power_pad_vrange_t

pad voltage range configuration.

typedef enum *_power_lvd_falling_trip_vol_val* power_lvd_falling_trip_vol_val_t

LVD falling trip voltage value.

typedef enum *_power_part_temp_range* power_part_temp_range_t

Part temperature range.

typedef enum *_power_volt_op_range* power_volt_op_range_t

Voltage operation range.

typedef enum *_power_vddcore_src* power_vddcore_src_t

VDDCORE supply source.

typedef enum *_power_control_for_pmic_mode* power_control_for_pmic_mode

vddcore or vdd1v8 power on selection for different PMIC mode. vddcore and vdd1v8 are always on in mode0. Refer to PMC->PMICCFG.

typedef void (*power_vddcore_set_func_t)(uint32_t millivolt)

Callback function used to change VDDCORE when the VDDCORE is supplied by external PMIC. Refer to *POWER_SetVddCoreSupplySrc()*

void POWER_PmicPowerModeSelectControl(uint32_t vddSelect)

API to set vddcore or vdd1v8 power on for PMIC modes which is responded to PDRUNCFG0[PMIC_MODE] or PDSLEEPCFG0[PMIC_MODE] select pin values. If not set, the driver will use default configurations for different PMIC mode. The default configuration is as below. PMIC_MODE: power mode select 0b00 run mode, all supplies on. 0b01 deep sleep mode, all supplies on. 0b10 deep powerdown mode, vddcore off. 0b11 full deep powerdown mode vdd1v8 and vddcore off.

Note, be cautious to modify the VDD state in different PMIC mode. When the default configuration is changed, use exclude_from_pd[0] to configure the PMIC mode for deep sleep and power down mode.

Parameters

- vddSelect – the ORd value of *power_control_for_pmic_mode*. Defines run (all supplies on), deep power-down (VDDCORE off), and true deep power-down (VDD1V8 and VDDCORE off).

static inline void POWER_EnablePD(*pd_bit_t* en)

API to enable PDRUNCFG bit in the Sysctl0. Note that enabling the bit powers down the peripheral.

Parameters

- en – peripheral for which to enable the PDRUNCFG bit

static inline void POWER_DisablePD(*pd_bit_t* en)

API to disable PDRUNCFG bit in the Sysctl0. Note that disabling the bit powers up the peripheral.

Parameters

- en – peripheral for which to disable the PDRUNCFG bit

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_UpdateOscSettlingTime(uint32_t osc_delay)

API to update XTAL oscillator settling time .

Parameters

- osc_delay – : OSC stabilization time in unit of microsecond

void POWER_UpdatePmicRecoveryTime(uint32_t pmic_delay)

API to update on-board PMIC vddcore recovery time.

NOTE: If LDO is used instead of PMIC, don't call it. Otherwise it must be called to allow power library to well handle the deep sleep process.

Parameters

- pmic_delay – : PMIC stabilization time in unit of microsecond, or PMIC_VDDCORE_RECOVERY_TIME_IGNORE if not care.

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 of interrupts to enable.

void POWER_DisableInterrupts(uint32_t interruptMask)

Disable the PMC interrupt requests.

Parameters

- interruptMask – : A bitmask of of interrupts to disable.

void POWER_SetAnalogBuffer(bool enable)

Set the PMC analog buffer for references or ATX2.

Parameters

- enable – : Set to true to enable analog buffer for references or ATX2, false to disable.

static inline 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

static inline *body_bias_mode_t* POWER_GetBodyBiasMode(*pmic_mode_reg_t* reg)

Get RBB/FBB bit value.

Parameters

- reg – : PDSLEEPCFG0 or PDRUNCFG0 register offset

Returns

Current body bias mode

void POWER_SetPadVolRange(const *power_pad_vrange_t* *config)

Configure pad voltage level. Wide voltage range cost more power due to enabled voltage detector.

NOTE: BE CAUTIOUS TO CALL THIS API. IF THE PAD SUPPLY IS BEYOND THE SET RANGE, SILICON MIGHT BE DAMAGED.

Parameters

- config – pad voltage range configuration.

void POWER_EnterRbb(void)

PMC Enter Rbb mode function call.

void POWER_EnterFbb(void)

PMC Enter Fbb mode function call.

void POWER_EnterNbb(void)

PMC exit Rbb & Fbb mode function call.

bool POWER_SetLdoVoltageForFreq(*power_part_temp_range_t* tempRange, *power_volt_op_range_t* voltOpRange, uint32_t cm33Freq, uint32_t dspFreq)

Deprecated and replaced by *POWER_SetVoltageForFreq()*! PMC Set Ldo volatage for particular frequency. NOTE: The API is only valid when MAINPLLCLKDIV[7:0] and DSPPLLCLKDIV[7:0] are 0. If LVD falling trip voltage is higher than the required core voltage for particular frequency, LVD voltage will be decreased to safe level to avoid unexpected LVD reset or interrupt event.

Parameters

- tempRange – : part temperature range
- voltOpRange – : voltage operation range.
- cm33Freq – : CM33 CPU clock frequency value
- dspFreq – : DSP CPU clock frequency value

Returns

true for success and false for CPU frequency out of specified voltOpRange.

void POWER_SetVddCoreSupplySrc(*power_vddcore_src_t* src)

Set VDDCORE supply source, PMIC or on-chip regulator.

Parameters

- src – : *power_vddcore_src_t*, VDDCore supply source

void POWER_SetPmicCoreSupplyFunc(*power_vddcore_set_func_t* func)

Set the core supply setting function if PMIC is used. The function is not needed and ignored when using the onchip regulator to supply VDDCORE.

Parameters

- func – : *power_vddcore_set_func_t*, the PMIC core supply voltage set function.

bool POWER_SetVoltageForFreq(*power_part_temp_range_t* tempRange, *power_volt_op_range_t* voltOpRange, uint32_t cm33Freq, uint32_t dspFreq, uint32_t mini_volt)

PMC Set volatage for particular frequency with given minimum value. *POWER_SetVddCoreSupplySrc* should be called in advance to tell power driver the supply source. If PMIC is used, the VDDCORE setting function should be set by *POWER_SetPmicCoreSupplyFunc* before this API is called. NOTE: The API is only valid when MAINPLLCLKDIV[7:0] and DSPPLLCLKDIV[7:0] are 0. If LVD falling trip voltage is higher than therequired core voltage for particular frequency, LVD voltage will be decreased to safe level to avoid unexpected LVD reset or interrupt event.

Parameters

- tempRange – : part temperature range
- voltOpRange – : voltage operation range.
- cm33Freq – : CM33 CPU clock frequency value
- dspFreq – : DSP CPU clock frequency value
- mini_volt – : minimum voltage in millivolt(mV) for VDDCORE. Should <= 1130mV, 0 means use the core frequency to calculate voltage.

Returns

true for success and false for CPU frequency out of specified voltOpRange or failed to set voltage.

void POWER_SetLvdFallingTripVoltage(*power_lvd_falling_trip_vol_val_t* volt)

Set vddcore low voltage detection falling trip voltage.

Parameters

- volt – target LVD voltage to set.

power_lvd_falling_trip_vol_val_t POWER_GetLvdFallingTripVoltage(void)

Get current vddcore low voltage detection falling trip voltage.

Returns

Current LVD voltage.

void POWER_DisableLVD(void)

Disable low voltage detection, no reset or interrupt is triggered when vddcore voltage drops below threshold. NOTE: This API is for internal use only. Application should not touch it.

void POWER_RestoreLVD(void)

Restore low voltage detection setting. NOTE: This API is for internal use only. Application should not touch it.

void POWER_SetPmicMode(uint32_t mode, *pmic_mode_reg_t* reg)

Set PMIC_MODE pins configure value.

Parameters

- mode – : PMIC MODE pin value
- reg – : PDSLEEPCFG0 or PDRUNCFG0 register offset

Returns

PMIC_MODE pins value in PDSLEEPCFG0

void POWER_EnterSleep(void)

Configures and enters in SLEEP low power mode.

void POWER_EnterDeepSleep(const uint32_t exclude_from_pd[4])

PMC Deep Sleep function call.

Parameters

- exclude_from_pd – Bit mask of the PDRUNCFG0 ~ PDRUNCFG3 that needs to be powered on during Deep Sleep mode selected.

void POWER_EnterDeepPowerDown(const uint32_t exclude_from_pd[4])

PMC Deep Power Down function call.

Parameters

- exclude_from_pd – Bit mask of the PDRUNCFG0 ~ PDRUNCFG3 that needs to be powered on during Deep Power Down mode selected.

void POWER_EnterFullDeepPowerDown(const uint32_t exclude_from_pd[4])

PMC Full Deep Power Down function call.

Parameters

- exclude_from_pd – Bit mask of the PDRUNCFG0 ~ PDRUNCFG3 that needs to be powered on during Full Deep Power Down mode selected.

void POWER_EnterPowerMode(*power_mode_cfg_t* mode, const uint32_t exclude_from_pd[4])

Power Library API to enter different power mode.

Parameters

- mode – Power mode to enter.
- exclude_from_pd – Bit mask of the PDRUNCFG0 ~ PDRUNCFG3 that needs to be powered on during power mode selected.

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.

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_DEFAULT_PMICMODE_DS

Define the default PMIC modes for power modes. NOTE: When using PMIC mode 0 (default mode) for low-power operation, the supply voltage must be carefully considered, since a CPU reset may not cause the PMIC mode to change.

POWER_DEFAULT_PMICMODE_DPD

POWER_DEFAULT_PMICMODE_FDPD

MAKE_PD_BITS(reg, slot)

SYSCTL0_PDRCFGSET_REG(x)

SYSCTL0_PDRCFGCLR_REG(x)

PDRCFG0

PDRCFG1

PDRCFG2

PDRCFG3

PMIC_VDDCORE_RECOVERY_TIME_IGNORE

PMIC is used but vddcore supply is always above LVD threshold.

SYSCTL0_TUPLE_REG(reg)

PMIC mode pin configuration API parameter.

uint32_t Vdde0Range

VDDE0 voltage range for VDDIO_0. *power_pad_vrange_val_t*

uint32_t Vdde1Range

VDDE1 voltage range for VDDIO_1. *power_pad_vrange_val_t*

uint32_t Vdde2Range

VDDE2 voltage range for VDDIO_2. *power_pad_vrange_val_t*

uint32_t ___pad0___

Reserved.

struct __power_pad_vrange

#include <fsl_power.h> pad voltage range configuration.

2.54 POWERQUAD: PowerQuad hardware accelerator

void PQ_GetDefaultConfig(*pq_config_t* *config)

Get default configuration.

This function initializes the POWERQUAD configuration structure to a default value. FORMAT register field definitions Bits[15:8] scaler (for scaled 'q31' formats) Bits[5:4] external format. 00b=q15, 01b=q31, 10b=float Bits[1:0] internal format. 00b=q15, 01b=q31, 10b=float POWERQUAD->INAFORMAT = (config->inputAPrescale « 8U) | (config->inputAFormat « 4U) | config->machineFormat

For all Powerquad operations internal format must be float (with the only exception being the FFT related functions, ie FFT/IFFT/DCT/IDCT which must be set to q31). The default values are: config->inputAFormat = kPQ_Float; config->inputAPrescale = 0; config->inputBFormat = kPQ_Float; config->inputBPrescale = 0; config->outputFormat = kPQ_Float; config->outputPrescale = 0; config->tmpFormat = kPQ_Float; config->tmpPrescale = 0; config->machineFormat = kPQ_Float; config->tmpBase = 0xE0000000;

Parameters

- config – Pointer to "pq_config_t" structure.

void PQ_SetConfig(POWERQUAD_Type *base, const *pq_config_t* *config)

Set configuration with format/prescale.

Parameters

- base – POWERQUAD peripheral base address
- config – Pointer to "pq_config_t" structure.

static inline void PQ_SetCoprocessorScaler(POWERQUAD_Type *base, const *pq_prescale_t* *prescale)

set coprocessor scaler for coprocessor instructions, this function is used to set output saturation and scaleing for input/output.

Parameters

- base – POWERQUAD peripheral base address
- prescale – Pointer to "pq_prescale_t" structure.

void PQ_Init(POWERQUAD_Type *base)

Initializes the POWERQUAD module.

Parameters

- base – POWERQUAD peripheral base address.

void PQ_Deinit(POWERQUAD_Type *base)

De-initializes the POWERQUAD module.

Parameters

- base – POWERQUAD peripheral base address.

void PQ_SetFormat(POWERQUAD_Type *base, *pq_computationengine_t* engine, *pq_format_t* format)

Set format for non-coprecessor instructions.

Parameters

- base – POWERQUAD peripheral base address
- engine – Computation engine
- format – Data format

static inline void PQ_WaitDone(POWERQUAD_Type *base)

Wait for the completion.

Parameters

- base – POWERQUAD peripheral base address

static inline void PQ_LnF32(float *pSrc, float *pDst)

Processing function for the floating-point natural log.

Parameters

- *pSrc – points to the block of input data. The range of the input value is (0 +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_InvF32(float *pSrc, float *pDst)

Processing function for the floating-point reciprocal.

Parameters

- *pSrc – points to the block of input data. The range of the input value is non-zero.
- *pDst – points to the block of output data

static inline void PQ_SqrtF32(float *pSrc, float *pDst)

Processing function for the floating-point square-root.

Parameters

- *pSrc – points to the block of input data. The range of the input value is [0 +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_InvSqrtF32(float *pSrc, float *pDst)

Processing function for the floating-point inverse square-root.

Parameters

- *pSrc – points to the block of input data. The range of the input value is (0 +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_EtoxF32(float *pSrc, float *pDst)

Processing function for the floating-point natural exponent.

Parameters

- *pSrc – points to the block of input data. The range of the input value is (-INFINITY +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_EtonxF32(float *pSrc, float *pDst)

Processing function for the floating-point natural exponent with negative parameter.

Parameters

- *pSrc – points to the block of input data. The range of the input value is (-INFINITY +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_SinF32(float *pSrc, float *pDst)

Processing function for the floating-point sine.

Parameters

- *pSrc – points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_CosF32(float *pSrc, float *pDst)

Processing function for the floating-point cosine.

Parameters

- *pSrc – points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
- *pDst – points to the block of output data

static inline void PQ_BiquadF32(float *pSrc, float *pDst)

Processing function for the floating-point biquad.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data

static inline void PQ_DivF32(float *x1, float *x2, float *pDst)

Processing function for the floating-point division.

Get x1 / x2.

Parameters

- x1 – x1
- x2 – x2
- *pDst – points to the block of output data

static inline void PQ_Biquad1F32(float *pSrc, float *pDst)

Processing function for the floating-point biquad.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data

static inline int32_t PQ_LnFixed(int32_t val)

Processing function for the fixed natural log.

Parameters

- val – value to be calculated. The range of the input value is (0 +INFINITY).

Returns

returns ln(val).

static inline int32_t PQ_InvFixed(int32_t val)

Processing function for the fixed reciprocal.

Parameters

- val – value to be calculated. The range of the input value is non-zero.

Returns

returns inv(val).

static inline uint32_t PQ_SqrtFixed(uint32_t val)

Processing function for the fixed square-root.

Parameters

- val – value to be calculated. The range of the input value is [0 +INFINITY).

Returns

returns sqrt(val).

static inline int32_t PQ_InvSqrtFixed(int32_t val)

Processing function for the fixed inverse square-root.

Parameters

- val – value to be calculated. The range of the input value is (0 +INFINITY).

Returns

returns 1/sqrt(val).

static inline int32_t PQ_EtoxFixed(int32_t val)

Processing function for the Fixed natural exponent.

Parameters

- val – value to be calculated. The range of the input value is (-INFINITY +INFINITY).

Returns

returns etox^(val).

static inline int32_t PQ_EtonxFixed(int32_t val)

Processing function for the fixed natural exponent with negative parameter.

Parameters

- val – value to be calculated. The range of the input value is (-INFINITY +INFINITY).

Returns

returns etonx^(val).

static inline int32_t PQ_SinQ31(int32_t val)

Processing function for the fixed sine.

Parameters

- val – value to be calculated. The input value is [-1, 1] in Q31 format, which means [-pi, pi].

Returns

returns sin(val).

static inline int16_t PQ_SinQ15(int16_t val)

Processing function for the fixed sine.

Parameters

- val – value to be calculated. The input value is [-1, 1] in Q15 format, which means [-pi, pi].

Returns

returns sin(val).

static inline int32_t PQ_CosQ31(int32_t val)

Processing function for the fixed cosine.

Parameters

- val – value to be calculated. The input value is [-1, 1] in Q31 format, which means [-pi, pi].

Returns
returns cos(val).

static inline int16_t PQ_CosQ15(int16_t val)

Processing function for the fixed sine.

Parameters

- val – value to be calculated. The input value is [-1, 1] in Q15 format, which means [-pi, pi].

Returns
returns sin(val).

static inline int32_t PQ_BiquadFixed(int32_t val)

Processing function for the fixed biquad.

Parameters

- val – value to be calculated

Returns
returns biquad(val).

void PQ_VectorLnF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised natural log.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised reciprocal.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSqrtF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvSqrtF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised inverse square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtoxF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised natural exponent.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtonxF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised natural exponent with negative parameter.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSinF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised sine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorCosF32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised cosine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorLnFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the Q31 vectorised natural log.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the Q31 vectorised reciprocal.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSqrtFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvSqrtFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised inverse square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtoxFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised natural exponent.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtonxFixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised natural exponent with negative parameter.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSinQ15(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the Q15 vectorised sine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorCosQ15(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the Q15 vectorised cosine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSinQ31(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the Q31 vectorised sine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorCosQ31(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the Q31 vectorised cosine.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorLnFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised natural log.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised reciprocal.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorSqrtFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorInvSqrtFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised inverse square-root.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtoxFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised natural exponent.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorEtonxFixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised natural exponent with negative parameter.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block of input data.

void PQ_VectorBiquadDf2F32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data.

void PQ_VectorBiquadDf2Fixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

void PQ_VectorBiquadDf2Fixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

void PQ_VectorBiquadCascadeDf2F32(float *pSrc, float *pDst, int32_t length)

Processing function for the floating-point vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

void PQ_VectorBiquadCascadeDf2Fixed32(int32_t *pSrc, int32_t *pDst, int32_t length)

Processing function for the 32-bit integer vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

void PQ_VectorBiquadCascadeDf2Fixed16(int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised biquad direct form II.

Parameters

- *pSrc – points to the block of input data
- *pDst – points to the block of output data
- length – the block size of input data

int32_t PQ_ArctanFixed(POWERQUAD_Type *base, int32_t x, int32_t y, *pq_cordic_iter_t* iteration)

Processing function for the fixed inverse trigonometric.

Get the inverse tangent, the behavior is like c function atan.

Note: The sum of x and y should not exceed the range of int32_t.

Note: Larger input number gets higher output accuracy, for example the arctan(0.5), the result of PQ_ArctanFixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24) is more accurate than PQ_ArctanFixed(POWERQUAD, 1, 2, kPQ_Iteration_24).

Parameters

- base – POWERQUAD peripheral base address
- x – value of opposite
- y – value of adjacent
- iteration – iteration times

Returns

The return value is in the range of -2^26 to 2^26, which means -pi/2 to pi/2.

int32_t PQ_ArctanhFixed(POWERQUAD_Type *base, int32_t x, int32_t y, *pq_cordic_iter_t* iteration)

Processing function for the fixed inverse trigonometric.

Note: The sum of x and y should not exceed the range of int32_t.

Note: Larger input number gets higher output accuracy, for example the arctanh(0.5), the result of PQ_ArctanhFixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24) is more accurate than PQ_ArctanhFixed(POWERQUAD, 1, 2, kPQ_Iteration_24).

Parameters

- base – POWERQUAD peripheral base address
- x – value of opposite
- y – value of adjacent
- iteration – iteration times

Returns

The return value is radians, 2^27 means pi. The range is -1.118 to 1.118 radians.

int32_t PQ_Arctan2Fixed(POWERQUAD_Type *base, int32_t x, int32_t y, *pq_cordic_iter_t* iteration)

Processing function for the fixed inverse trigonometric.

Get the inverse tangent, it calculates the angle in radians for the quadrant. The behavior is like c function atan2.

Note: The sum of x and y should not exceed the range of int32_t.

Note: Larger input number gets higher output accuracy, for example the arctan(0.5), the result of PQ_Arctan2Fixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24) is more accurate than PQ_Arctan2Fixed(POWERQUAD, 1, 2, kPQ_Iteration_24).

Parameters

- base – POWERQUAD peripheral base address
- x – value of opposite
- y – value of adjacent
- iteration – iteration times

Returns

The return value is in the range of -2^27 to 2^27, which means -pi to pi.

static inline int32_t PQ_Biquad1Fixed(int32_t val)

Processing function for the fixed biquad.

Parameters

- val – value to be calculated

Returns

returns biquad(val).

void PQ_TransformCFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the complex FFT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

void PQ_TransformRFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the real FFT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

void PQ_TransformIFFT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the inverse complex FFT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples

- pData – input data
- pResult – output data.

void PQ_TransformCDCT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the complex DCT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

void PQ_TransformRDCT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the real DCT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

void PQ_TransformIDCT(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the inverse complex DCT.

Parameters

- base – POWERQUAD peripheral base address
- length – number of input samples
- pData – input data
- pResult – output data.

void PQ_BiquadBackUpInternalState(POWERQUAD_Type *base, int32_t biquad_num, *pq_biquad_state_t* *state)

Processing function for backup biquad context.

Parameters

- base – POWERQUAD peripheral base address
- biquad_num – biquad side
- state – point to states.

void PQ_BiquadRestoreInternalState(POWERQUAD_Type *base, int32_t biquad_num, *pq_biquad_state_t* *state)

Processing function for restore biquad context.

Parameters

- base – POWERQUAD peripheral base address
- biquad_num – biquad side
- state – point to states.

void PQ_BiquadCascadeDf2Init(*pq_biquad_cascade_df2_instance* *S, uint8_t numStages, *pq_biquad_state_t* *pState)

Initialization function for the direct form II Biquad cascade filter.

Parameters

- *S – **[inout]** points to an instance of the filter data structure.
- numStages – **[in]** number of 2nd order stages in the filter.
- *pState – **[in]** points to the state buffer.

void PQ_BiquadCascadeDf2F32(const *pq_biquad_cascade_df2_instance* *S, float *pSrc, float *pDst, uint32_t blockSize)

Processing function for the floating-point direct form II Biquad cascade filter.

Parameters

- *S – **[in]** points to an instance of the filter data structure.
- *pSrc – **[in]** points to the block of input data.
- *pDst – **[out]** points to the block of output data
- blockSize – **[in]** number of samples to process.

void PQ_BiquadCascadeDf2Fixed32(const *pq_biquad_cascade_df2_instance* *S, int32_t *pSrc, int32_t *pDst, uint32_t blockSize)

Processing function for the Q31 direct form II Biquad cascade filter.

Parameters

- *S – **[in]** points to an instance of the filter data structure.
- *pSrc – **[in]** points to the block of input data.
- *pDst – **[out]** points to the block of output data
- blockSize – **[in]** number of samples to process.

void PQ_BiquadCascadeDf2Fixed16(const *pq_biquad_cascade_df2_instance* *S, int16_t *pSrc, int16_t *pDst, uint32_t blockSize)

Processing function for the Q15 direct form II Biquad cascade filter.

Parameters

- *S – **[in]** points to an instance of the filter data structure.
- *pSrc – **[in]** points to the block of input data.
- *pDst – **[out]** points to the block of output data
- blockSize – **[in]** number of samples to process.

void PQ_FIR(POWERQUAD_Type *base, const void *pAData, uint16_t ALength, const void *pBData, uint16_t BLength, void *pResult, uint32_t opType)

Processing function for the FIR.

Parameters

- base – POWERQUAD peripheral base address
- pAData – the first input sequence
- ALength – number of the first input sequence
- pBData – the second input sequence
- BLength – number of the second input sequence
- pResult – array for the output data

- opType – operation type, could be PQ_FIR_FIR, PQ_FIR_CONVOLUTION, PQ_FIR_CORRELATION.

void PQ_FIRIncrement(POWERQUAD_Type *base, uint16_t ALength, uint16_t BLength, int32_t xOffset)

Processing function for the incremental FIR. This function can be used after pq_fir() for incremental FIR operation when new x data are available.

Parameters

- base – POWERQUAD peripheral base address
- ALength – number of input samples
- BLength – number of taps
- xOffset – offset for number of input samples

void PQ_MatrixAddition(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

Processing function for the matrix addition.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro *POWERQUAD_MAKE_MATRIX_LEN*.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

void PQ_MatrixSubtraction(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

Processing function for the matrix subtraction.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro *POWERQUAD_MAKE_MATRIX_LEN*.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

void PQ_MatrixMultiplication(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

Processing function for the matrix multiplication.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro *POWERQUAD_MAKE_MATRIX_LEN*.

- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

void PQ_MatrixProduct(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

Processing function for the matrix product.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro *POWERQUAD_MAKE_MATRIX_LEN*.
- pAData – input matrix A
- pBData – input matrix B
- pResult – array for the output data.

void PQ_VectorDotProduct(POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

Processing function for the vector dot product.

Parameters

- base – POWERQUAD peripheral base address
- length – length of vector
- pAData – input vector A
- pBData – input vector B
- pResult – array for the output data.

void PQ_MatrixInversion(POWERQUAD_Type *base, uint32_t length, void *pData, void *pTmpData, void *pResult)

Processing function for the matrix inverse.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro *POWERQUAD_MAKE_MATRIX_LEN*.
- pData – input matrix
- pTmpData – input temporary matrix, pTmpData length not less than pData lenght and 1024 words is sufficient for the largest supported matrix.
- pResult – array for the output data, round down for fixed point.

void PQ_MatrixTranspose(POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

Processing function for the matrix transpose.

Parameters

- base – POWERQUAD peripheral base address

- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro *POWERQUAD_MAKE_MATRIX_LEN*.
- pData – input matrix
- pResult – array for the output data.

void PQ_MatrixScale(POWERQUAD_Type *base, uint32_t length, float misc, const void *pData, void *pResult)

Processing function for the matrix scale.

Parameters

- base – POWERQUAD peripheral base address
- length – rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro *POWERQUAD_MAKE_MATRIX_LEN*.
- misc – scaling parameters
- pData – input matrix
- pResult – array for the output data.

FSL_POWERQUAD_DRIVER_VERSION

Version.

enum pq_computationengine_t

powerquad computation engine

Values:

enumerator kPQ_CP_PQ

Math engine.

enumerator kPQ_CP_MTX

Matrix engine.

enumerator kPQ_CP_FFT

FFT engine.

enumerator kPQ_CP_FIR

FIR engine.

enumerator kPQ_CP_CORDIC

CORDIC engine.

enum pq_format_t

powerquad data structure format type

Values:

enumerator kPQ_16Bit

Int16 Fixed point.

enumerator kPQ_32Bit
Int32 Fixed point.

enumerator kPQ_Float
Float point.

enum pq_cordic_iter_t
CORDIC iteration.
Values:

enumerator kPQ_Iteration_8
Iterate 8 times.

enumerator kPQ_Iteration_16
Iterate 16 times.

enumerator kPQ_Iteration_24
Iterate 24 times.

typedef struct *_pq_biquad_param* pq_biquad_param_t
Struct to save biquad parameters.

typedef struct *_pq_biquad_state* pq_biquad_state_t
Struct to save biquad state.

typedef union *_pq_float* pq_float_t
Conversion between integer and float type.

PQ_VectorBiqaudDf2F32

PQ_VectorBiqaudDf2Fixed32

PQ_VectorBiqaudDf2Fixed16

PQ_VectorBiqaudCascadeDf2F32

PQ_VectorBiqaudCascadeDf2Fixed32

PQ_VectorBiqaudCascadeDf2Fixed16

PQ_Vector8BiqaudDf2CascadeF32

PQ_Vector8BiqaudDf2CascadeFixed32

PQ_Vector8BiqaudDf2CascadeFixed16

PQ_FLOAT32

PQ_FIXEDPT

CP_PQ

CP_MTX

CP_FFT

CP_FIR

CP_CORDIC

PQ_TRANS

PQ_TRIG

PQ_BIQUAD

PQ_TRANS_FIXED

PQ_TRIG_FIXED

PQ_BIQUAD_FIXED

PQ_INV

PQ_LN

PQ_SQRT

PQ_INVSQRT

PQ_ETOX

PQ_ETONX

PQ_DIV

PQ_SIN

PQ_COS

PQ_BIQ0_CALC

PQ_BIQ1_CALC

PQ_COMP0_ONLY

PQ_COMP1_ONLY

CORDIC_ITER(x)

CORDIC_MIU(x)

CORDIC_T(x)

CORDIC_ARCTAN

CORDIC_ARCTANH

INST_BUSY

PQ_ERRSTAT_OVERFLOW

PQ_ERRSTAT_NAN

PQ_ERRSTAT_FIXEDOVERFLOW

PQ_ERRSTAT_UNDERFLOW

PQ_TRANS_CFFT

PQ_TRANS_IFFT

PQ_TRANS_CDCT

PQ_TRANS_IDCT

PQ_TRANS_RFFT

PQ_TRANS_RDCT

PQ_MTX_SCALE

PQ_MTX_MULT

PQ_MTX_ADD

PQ_MTX_INV

PQ_MTX_PROD

PQ_MTX_SUB

PQ_VEC_DOTP

PQ_MTX_TRAN

PQ_FIR_FIR

PQ_FIR_CONVOLUTION

PQ_FIR_CORRELATION

PQ_FIR_INCREMENTAL

_pq_ln0(**x**)

_pq_inv0(**x**)

_pq_sqrt0(**x**)

_pq_invsqrt0(**x**)

_pq_etox0(**x**)

_pq_etonx0(**x**)

_pq_sin0(**x**)

_pq_cos0(**x**)

_pq_biquad0(**x**)

_pq_ln_fx0(**x**)

_pq_inv_fx0(**x**)

_pq_sqrt_fx0(**x**)

_pq_invsqrt_fx0(**x**)

_pq_etox_fx0(**x**)

_pq_etonx_fx0(**x**)

_pq_sin_fx0(**x**)

_pq_cos_fx0(**x**)

_pq_biquad0_fx(**x**)

_pq_div0(**x**)

_pq_div1(**x**)

_pq_ln1(**x**)

_pq_inv1(**x**)

_pq_sqrt1(**x**)

_pq_invsqrt1(**x**)

__pq__etox1(**x**)

__pq__etonx1(**x**)

__pq__sin1(**x**)

__pq__cos1(**x**)

__pq__biquad1(**x**)

__pq__ln__fx1(**x**)

__pq__inv__fx1(**x**)

__pq__sqrt__fx1(**x**)

__pq__invsqrt__fx1(**x**)

__pq__etox__fx1(**x**)

__pq__etonx__fx1(**x**)

__pq__sin__fx1(**x**)

__pq__cos__fx1(**x**)

__pq__biquad1__fx(**x**)

__pq__readMult0()

__pq__readAdd0()

__pq__readMult1()

__pq__readAdd1()

__pq__readMult0__fx()

__pq__readAdd0__fx()

__pq__readMult1__fx()

__pq__readAdd1__fx()

PQ__LN__INF
: Parameter used for vector ln(x)

PQ__INV__INF
: Parameter used for vector 1/x

PQ__SQRT__INF
: Parameter used for vector sqrt(x)

PQ__ISQRT__INF
: Parameter used for vector 1/sqrt(x)

PQ__ETOX__INF
: Parameter used for vector e^x

PQ_ETONX_INF
Parameter used for vector e^(-x)

PQ_SIN_INF
Parameter used for vector sin(x)

PQ_COS_INF
Parameter used for vector cos(x)

PQ_RUN_OPCODE_R3_R2(BATCH_OPCODE, BATCH_MACHINE)

PQ_RUN_OPCODE_R5_R4(BATCH_OPCODE, BATCH_MACHINE)

PQ_RUN_OPCODE_R7_R6(BATCH_OPCODE, BATCH_MACHINE)

PQ_Vector8_FP(middle, last, BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

PQ_RUN_OPCODE_R2_R3(BATCH_OPCODE, BATCH_MACHINE)

PQ_RUN_OPCODE_R4_R5(BATCH_OPCODE, BATCH_MACHINE)

PQ_RUN_OPCODE_R6_R7(BATCH_OPCODE, BATCH_MACHINE)

PQ_Vector8_FX(middle, last, BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

PQ_Initiate_Vector_Func(pSrc, pDst)
Start 32-bit data vector calculation.

Start the vector calculation, the input data could be float, int32_t or Q31.

Parameters

- pSrc – Pointer to the source data.
- pDst – Pointer to the destination data.

PQ_End_Vector_Func()
End vector calculation.

This function should be called after vector calculation.

PQ_StartVector(PSRC, PDST, LENGTH)
Start 32-bit data vector calculation.

Start the vector calculation, the input data could be float, int32_t or Q31.

Parameters

- PSRC – Pointer to the source data.
- PDST – Pointer to the destination data.
- LENGTH – Number of the data, must be multiple of 8.

PQ_StartVectorFixed16(PSRC, PDST, LENGTH)
Start 16-bit data vector calculation.

Start the vector calculation, the input data could be int16_t. This function should be use with *PQ_Vector8Fixed16*.

Parameters

- PSRC – Pointer to the source data.
- PDST – Pointer to the destination data.
- LENGTH – Number of the data, must be multiple of 8.

PQ_StartVectorQ15(PSRC, PDST, LENGTH)

Start Q15-bit data vector calculation.

Start the vector calculation, the input data could be Q15. This function should be use with *PQ_Vector8Q15*. This function is dedicate for SinQ15/CosQ15 vector calculation. Because PowerQuad only supports Q31 Sin/Cos fixed function, so the input Q15 data is left shift 16 bits first, after Q31 calculation, the output data is right shift 16 bits.

Parameters

- PSRC – Pointer to the source data.
- PDST – Pointer to the destination data.
- LENGTH – Number of the data, must be multiple of 8.

PQ_EndVector()

End vector calculation.

This function should be called after vector calculation.

PQ_Vector8F32(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

Float data vector calculation.

Float data vector calculation, the input data should be float. The parameter could be PQ_LN_INF, PQ_INV_INF, PQ_SQRT_INF, PQ_ISQRT_INF, PQ_ETOX_INF, PQ_ETONX_INF. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
float output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

PQ_Vector8Fixed32(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

Fixed 32bits data vector calculation.

Float data vector calculation, the input data should be 32-bit integer. The parameter could be PQ_LN_INF, PQ_INV_INF, PQ_SQRT_INF, PQ_ISQRT_INF, PQ_ETOX_INF, PQ_ETONX_INF. PQ_SIN_INF, PQ_COS_INF. When this function is used for sin/cos calculation, the input data should be in the format Q1.31. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 4, 9, 16, 25, 36, 49, 64};
int32_t output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

PQ_Vector8Fixed16(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

Fixed 32bits data vector calculation.

Float data vector calculation, the input data should be 16-bit integer. The parameter could be PQ_LN_INF, PQ_INV_INF, PQ_SQRT_INF, PQ_ISQRT_INF, PQ_ETOX_INF, PQ_ETONX_INF. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {1, 4, 9, 16, 25, 36, 49, 64};
int16_t output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
```

(continues on next page)

(continued from previous page)

```
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

PQ_Vector8Q15(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)

Q15 data vector calculation.

Q15 data vector calculation, this function should only be used for sin/cos Q15 calculation, and the coprocessor output prescaler must be set to 31 before this function. This function loads Q15 data and left shift 16 bits, calculate and right shift 16 bits, then stores to the output array. The input range -1 to 1 means -pi to pi. For example, to calculate sin of a vector, use like this:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {...}
int16_t output[VECTOR_LEN];
const pq_prescale_t prescale =
{
    .inputPrescale = 0,
    .outputPrescale = 31,
    .outputSaturate = 0
};

PQ_SetCoprocessorScaler(POWERQUAD, const pq_prescale_t *prescale);

PQ_StartVectorQ15(pSrc, pDst, length);
PQ_Vector8Q15(PQ_SQRT_INF);
PQ_EndVector();
```

PQ_DF2_Vector8_FP(middle, last)

Float data vector biquad direct form II calculation.

Biquad filter, the input and output data are float data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 16
float input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_Initiate_Vector_Func(pSrc,pDst);
PQ_DF2_Vector8_FP(false,false);
PQ_DF2_Vector8_FP(true,true);
PQ_End_Vector_Func();
```

PQ_DF2_Vector8_FX(middle, last)

Fixed data vector biquad direct form II calculation.

Biquad filter, the input and output data are fixed data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 16
int32_t input[VECTOR_LEN] = {1024, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_Initiate_Vector_Func(pSrc,pDst);
PQ_DF2_Vector8_FX(false,false);
PQ_DF2_Vector8_FX(true,true);
PQ_End_Vector_Func();
```

PQ_Vector8BiquadDf2F32()

Float data vector biquad direct form II calculation.

Biquad filter, the input and output data are float data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
float output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2F32();
PQ_EndVector();
```

PQ_Vector8BiquadDf2Fixed32()

Fixed 32-bit data vector biquad direct form II calculation.

Biquad filter, the input and output data are Q31 or 32-bit integer. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
```

(continues on next page)

(continued from previous page)

```
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2Fixed32();
PQ_EndVector();
```

PQ_Vector8BiquadDf2Fixed16()

Fixed 16-bit data vector biquad direct form II calculation.

Biquad filter, the input and output data are Q15 or 16-bit integer. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int16_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2Fixed16();
PQ_EndVector();
```

PQ_DF2_Cascade_Vector8_FP(middle, last)

Float data vector direct form II biquad cascade filter.

The input and output data are float data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 16
float input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};
```

(continues on next page)

(continued from previous page)

```
pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FP(false, false);
PQ_DF2_Cascade_Vector8_FP(true, true);
PQ_End_Vector_Func();
```

PQ_DF2_Cascade_Vector8_FX(middle, last)

Fixed data vector direct form II biquad cascade filter.

The input and output data are fixed data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 16
int32_t input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FX(false, false);
PQ_DF2_Cascade_Vector8_FX(true, true);
PQ_End_Vector_Func();
```

PQ_Vector8BiquadDf2CascadeF32()

Float data vector direct form II biquad cascade filter.

The input and output data are float data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
float output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeF32();
PQ_EndVector();
```

PQ_Vector8BiquadDf2CascadeFixed32()

Fixed 32-bit data vector direct form II biquad cascade filter.

The input and output data are fixed 32-bit data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
```

(continues on next page)

(continued from previous page)

```
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeFixed32();
PQ_EndVector();
```

PQ_Vector8BiquadDf2CascadeFixed16()

Fixed 16-bit data vector direct form II biquad cascade filter.

The input and output data are fixed 16-bit data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeFixed16();
PQ_EndVector();
```

POWERQUAD_MAKE_MATRIX_LEN(mat1Row, mat1Col, mat2Col)

Make the length used for matrix functions.

PQ_Q31_2_FLOAT(x)

Convert Q31 to float.

PQ_Q15_2_FLOAT(x)
Convert Q15 to float.

FSL_POWERQUAD_CMSIS_DRIVER_VERSION
Version.

pq_arm_cos_f32

pq_arm_cos_q31

pq_arm_cos_q15

pq_arm_sin_f32

pq_arm_sin_q31

pq_arm_sin_q15

pq_arm_sqrt_q31

pq_arm_sqrt_q15

pq_arm_cfft_q31

pq_arm_cfft_q15

pq_arm_rfft_init_q31

pq_arm_rfft_q31

pq_arm_rfft_init_q15

pq_arm_rfft_q15

pq_arm_dct4_init_q31

pq_arm_dct4_q31

pq_arm_dct4_init_q15

pq_arm_dct4_q15

pq_arm_fir_init_f32

pq_arm_fir_init_q31

pq_arm_fir_init_q15

pq_arm_fir_f32

pq_arm_fir_q31

pq_arm_fir_q15

pq_arm_conv_f32

pq_arm_conv_q31

pq_arm_conv_q15

pq_arm_correlate_f32

pq_arm_correlate_q31

pq_arm_correlate_q15

pq_arm_mat_init_f32

pq_arm_mat_init_q31

pq_arm_mat_init_q15

pq_arm_mat_add_f32

pq_arm_mat_add_q31

pq_arm_mat_add_q15

pq_arm_mat_sub_f32

pq_arm_mat_sub_q31

pq_arm_mat_sub_q15

pq_arm_mat_mult_f32

pq_arm_mat_mult_q31

pq_arm_mat_mult_q15

pq_arm_mat_inverse_f32

pq_arm_mat_trans_f32

pq_arm_mat_trans_q31

pq_arm_mat_trans_q15

pq_arm_mat_scale_f32

pq_arm_mat_scale_q31

pq_arm_mat_scale_q15

struct pq_prescale_t

#include <fsl_powerquad.h> Coprocessor prescale.

Public Members

int8_t inputPrescale

Input prescale.

int8_t outputPrescale

Output prescale.

int8_t outputSaturate

Output saturate at n bits, for example 0x11 is 8 bit space, the value will be truncated at +127 or -128.

struct pq_config_t

#include <fsl_powerquad.h> powerquad data structure format

Public Members

pq_format_t inputAFormat

Input A format.

int8_t inputAPrescale

Input A prescale, for example 1.5 can be 1.5*2^n if you scale by 'shifting' ('scaling' by a factor of n).

pq_format_t inputBFormat

Input B format.

int8_t inputBPrescale

Input B prescale.

pq_format_t outputFormat
Out format.

int8_t outputPrescale
Out prescale.

pq_format_t tmpFormat
Temp format.

int8_t tmpPrescale
Temp prescale.

pq_format_t machineFormat
Machine format.

uint32_t *tmpBase
Tmp base address.

struct __pq__biquad__param
#include <fsl_powerquad.h> Struct to save biquad parameters.

Public Members

float v__n__1
v[n-1], set to 0 when initialization.

float v__n
v[n], set to 0 when initialization.

float a__1
a[1]

float a__2
a[2]

float b__0
b[0]

float b__1
b[1]

float b__2
b[2]

struct __pq__biquad__state
#include <fsl_powerquad.h> Struct to save biquad state.

Public Members

pq_biquad_param_t param
Filter parameter.

uint32_t compreg
Internal register, set to 0 when initialization.

struct pq_biquad_cascade_df2_instance
#include <fsl_powerquad.h> Instance structure for the direct form II Biquad cascade filter.

Public Members

uint8_t numStages
Number of 2nd order stages in the filter.

pq_biquad_state_t *pState
Points to the array of state coefficients.

union _pq_float
#include <fsl_powerquad.h> Conversion between integer and float type.

Public Members

float floatX
Float type.

uint32_t integerX
Unsigned interger type.

2.55 PUF: Physical Unclonable Function

FSL_PUF_DRIVER_VERSION
PUF driver version. Version 2.2.1.

Current version: 2.2.1

Change log:

- 2.0.0
  - Initial version.
- 2.0.1
  - Fixed puf_wait_usec function optimization issue.
- 2.0.2
  - Add PUF configuration structure and support for PUF SRAM controller. Remove magic constants.

- 2.0.3
  - Fix MISRA C-2012 issue.
- 2.1.0
  - Align driver with PUF SRAM controller registers on LPCXpresso55s16.
  - Update initizalition logic .
- 2.1.1
  - Fix ARMGCC build warning .
- 2.1.2
  - Update: Add automatic big to little endian swap for user (pre-shared) keys destinated to secret hardware bus (PUF key index 0).
- 2.1.3
  - Fix MISRA C-2012 issue.
- 2.1.4
  - Replace register uint32_t ticksCount with volatile uint32_t ticksCount in puf_wait_usec() to prevent optimization out delay loop.
- 2.1.5
  - Use common SDK delay in puf_wait_usec()
- 2.1.6
  - Changed wait time in PUF_Init(), when initialization fails it will try PUF_Powercycle() with shorter time. If this shorter time will also fail, initialization will be tried with worst case time as before.
- 2.2.0
- Add support for kPUF_KeySlot4.
- Add new PUF_ClearKey() function, that clears a desired PUF internal HW key register.
- 2.2.1
  - Fix CERT-C violations (ARR30-C/OVERRUN in PUF_GetHwKey, INT30-C in PUF_GetKey).

enum _puf_key_index_register

Values:

enumerator kPUF_KeyIndex_00

enumerator kPUF_KeyIndex_01

enumerator kPUF_KeyIndex_02

enumerator kPUF_KeyIndex_03

enumerator kPUF_KeyIndex_04

enumerator kPUF_KeyIndex_05

enumerator kPUF_KeyIndex_06

enumerator kPUF_KeyIndex_07

enumerator kPUF_KeyIndex_08

enumerator kPUF_KeyIndex_09

enumerator kPUF_KeyIndex_10

enumerator kPUF_KeyIndex_11

enumerator kPUF_KeyIndex_12

enumerator kPUF_KeyIndex_13

enumerator kPUF_KeyIndex_14

enumerator kPUF_KeyIndex_15

enum _puf_min_max

Values:

enumerator kPUF_KeySizeMin

enumerator kPUF_KeySizeMax

enumerator kPUF_KeyIndexMax

enum _puf_key_slot

PUF key slot.

Values:

enumerator kPUF_KeySlot0

PUF key slot 0

enumerator kPUF_KeySlot1

PUF key slot 1

PUF status return codes.

Values:

enumerator kStatus_EnrollNotAllowed

enumerator kStatus_StartNotAllowed

typedef enum *_puf_key_index_register* puf_key_index_register_t

typedef enum *_puf_min_max* puf_min_max_t

typedef enum *_puf_key_slot* puf_key_slot_t
PUF key slot.

PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(**x**)
Get Key Code size in bytes from key size in bytes at compile time.

PUF_MIN_KEY_CODE_SIZE

PUF_ACTIVATION_CODE_SIZE

KEYSTORE_PUF_DISCHARGE_TIME_FIRST_TRY_MS

KEYSTORE_PUF_DISCHARGE_TIME_MAX_MS

struct puf_config_t
#include <fsl_puf.h>

2.56 Reset Driver

enum _RSTCTL_RSTn
Enumeration for peripheral reset control bits.
Defines the enumeration for peripheral reset control bits in RSTCTLx registers
Values:

enumerator kDSP_RST_SHIFT_RSTn
DSP reset control

enumerator kPOWERQUAD_RST_SHIFT_RSTn
POWERQUAD reset control

enumerator kCASPER_RST_SHIFT_RSTn
CASPER reset control

enumerator kHASHCRYPT_RST_SHIFT_RSTn
HASHCRYPT reset control

enumerator kPUF_RST_SHIFT_RSTn
Physical unclonable function reset control

enumerator kRNG_RST_SHIFT_RSTn
Random number generator (RNG) reset control

enumerator kFLEXSPI_RST_SHIFT_RSTn
FLEXSPI reset control

enumerator kUSBHS_PHY_RST_SHIFT_RSTn
High speed USB PHY reset control

enumerator kUSBHS_DEVICE_RST_SHIFT_RSTn
High speed USB Device reset control

enumerator kUSBHS_HOST_RST_SHIFT_RSTn
High speed USB Host reset control

enumerator kUSBHS_SRAM_RST_SHIFT_RSTn
High speed USB SRAM reset control

enumerator kSCT_RST_SHIFT_RSTn
Standard ctimers reset control

enumerator kSDIO0_RST_SHIFT_RSTn
SDIO0 reset control

enumerator kSDIO1_RST_SHIFT_RSTn
SDIO1 reset control

enumerator kACMP0_RST_SHIFT_RSTn
Grouped interrupt (PINT) reset control.

enumerator kADC0_RST_SHIFT_RSTn
ADC0 reset control

enumerator kSHSGPIO0_RST_SHIFT_RSTn
Security HSGPIO 0 reset control

enumerator kUTICK0_RST_SHIFT_RSTn
Micro-tick timer reset control

enumerator kWWDT0_RST_SHIFT_RSTn
Windowed Watchdog timer 0 reset control

enumerator kFC0_RST_SHIFT_RSTn
Flexcomm Interface 0 reset control

enumerator kFC1_RST_SHIFT_RSTn
Flexcomm Interface 1 reset control

enumerator kFC2_RST_SHIFT_RSTn
Flexcomm Interface 2 reset control

enumerator kFC3_RST_SHIFT_RSTn
Flexcomm Interface 3 reset control

enumerator kFC4_RST_SHIFT_RSTn
Flexcomm Interface 4 reset control

enumerator kFC5_RST_SHIFT_RSTn
Flexcomm Interface 5 reset control

enumerator kFC6_RST_SHIFT_RSTn
Flexcomm Interface 6 reset control

enumerator kFC7_RST_SHIFT_RSTn
Flexcomm Interface 7 reset control

enumerator kFC14_RST_SHIFT_RSTn
Flexcomm Interface 14 reset control

enumerator kFC15_RST_SHIFT_RSTn
Flexcomm Interface 15 reset control

enumerator kDMIC_RST_SHIFT_RSTn
Digital microphone interface reset control

enumerator kOSEVENT_TIMER_RST_SHIFT_RSTn
Osevent Timer reset control

enumerator kHSGPIO0_RST_SHIFT_RSTn
HSGPIO 0 reset control

enumerator kHSGPIO1_RST_SHIFT_RSTn
HSGPIO 1 reset control

enumerator kHSGPIO2_RST_SHIFT_RSTn
HSGPIO 2 reset control

enumerator kHSGPIO3_RST_SHIFT_RSTn
HSGPIO 3 reset control

enumerator kHSGPIO4_RST_SHIFT_RSTn
HSGPIO 4 reset control

enumerator kHSGPIO5_RST_SHIFT_RSTn
HSGPIO 5 reset control

enumerator kHSGPIO6_RST_SHIFT_RSTn
HSGPIO 6 reset control

enumerator kHSGPIO7_RST_SHIFT_RSTn
HSGPIO 7 reset control

enumerator kCRC_RST_SHIFT_RSTn
CRC reset control

enumerator kDMAC0_RST_SHIFT_RSTn
DMA Controller 0 reset control

enumerator kDMAC1_RST_SHIFT_RSTn
DMA Controller 1 reset control

enumerator kMU_RST_SHIFT_RSTn
Message Unit reset control

enumerator kSEMA_RST_SHIFT_RSTn
Semaphore reset control

enumerator kFREQME_RST_SHIFT_RSTn
Frequency Measure reset control

enumerator kCT32B0_RST_SHIFT_RSTn
CT32B0 reset control

enumerator kCT32B1_RST_SHIFT_RSTn
CT32B1 reset control

enumerator kCT32B2_RST_SHIFT_RSTn
CT32B3 reset control

enumerator kCT32B3_RST_SHIFT_RSTn
CT32B4 reset control

enumerator kCT32B4_RST_SHIFT_RSTn
CT32B4 reset control

enumerator kMRT0_RST_SHIFT_RSTn
Multi-rate timer (MRT) reset control

enumerator kWWDT1_RST_SHIFT_RSTn
Windowed Watchdog timer 1 reset control

enumerator kI3C0_RST_SHIFT_RSTn
I3C reset control

enumerator kPINT_RST_SHIFT_RSTn
GPIO Pin interrupt reset control

enumerator kINPUTMUX_RST_SHIFT_RSTn

Peripheral input muxes reset control

typedef enum *_RSTCTL_RSTn* RSTCTL_RSTn_t

Enumeration for peripheral reset control bits.

Defines the enumeration for peripheral reset control bits in RSTCTLx registers

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.

FSL_RESET_DRIVER_VERSION

reset driver version 2.2.2.

RST_CTL0_PSCCTL0

Reset control registers index.

RST_CTL0_PSCCTL1

RST_CTL0_PSCCTL2

RST_CTL1_PSCCTL0

RST_CTL1_PSCCTL1

RST_CTL1_PSCCTL2

ACMP_RSTS
: Array initializers with peripheral reset bits

DSP_RSTS

FLEXSPI_RSTS

SEMA42_RSTS

ADC_RSTS

CASPER_RSTS

CRC_RSTS

DMA_RSTS_N

DMIC_RSTS

FLEXCOMM_RSTS

GPIO_RSTS_N

HASHCRYPT_RSTS

I3C_RSTS

INPUTMUX_RSTS

MRT_RSTS

MU_RSTS

PINT_RSTS

FREQME_RSTS

SCT_RSTS

CTIMER_RSTS

USB_RSTS

USDHC_RSTS

UTICK_RSTS

WWDT_RSTS

OSTIMER_RSTS

POWERQUAD_RSTS

PUF_RSTS

TRNG_RSTS

2.57 RTC: Real Time Clock

void RTC_Init(RTC_Type *base)

Un-gate the RTC clock and enable the RTC oscillator.

Note: This API should be called at the beginning of the application using the RTC driver.

Parameters

- base – RTC peripheral base address

static inline void RTC_Deinit(RTC_Type *base)

Stop the timer and gate the RTC clock.

Parameters

- base – RTC peripheral base address

status_t RTC_SetDatetime(RTC_Type *base, const *rtc_datetime_t* *datetime)

Set the RTC date and time according to the given time structure.

The RTC counter must be stopped prior to calling this function as writes to the RTC seconds register will fail if the RTC counter is running.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to structure where the date and time details to set are stored

Returns

kStatus_Success: Success in setting the time and starting the RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

void RTC_GetDatetime(RTC_Type *base, *rtc_datetime_t* *datetime)

Get the RTC time and stores it in the given time structure.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to structure where the date and time details are stored.

status_t RTC_SetAlarm(RTC_Type *base, const *rtc_datetime_t* *alarmTime)

Set the RTC alarm time.

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

- base – RTC peripheral base address
- alarmTime – Pointer to structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

void RTC_GetAlarm(RTC_Type *base, *rtc_datetime_t* *datetime)

Return the RTC alarm time.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to structure where the alarm date and time details are stored.

static inline void RTC_EnableWakeupTimer(RTC_Type *base, bool enable)

Enable the RTC wake-up timer (1KHZ).

After calling this function, the RTC driver will use/un-use the RTC wake-up (1KHZ) at the same time.

Parameters

- base – RTC peripheral base address
- enable – Use/Un-use the RTC wake-up timer.
  - 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 RTC_GetEnabledWakeupTimer(RTC_Type *base)

Get the enabled status of the RTC wake-up timer (1KHZ).

Parameters

- base – RTC peripheral base address

Returns

The enabled status of RTC wake-up timer (1KHZ).

static inline void RTC_EnableSubsecCounter(RTC_Type *base, bool enable)

Enable the RTC Sub-second counter (32KHZ).

Note: Only enable sub-second counter after RTC_ENA bit has been set to 1.

Parameters

- base – RTC peripheral base address
- enable – Enable/Disable RTC sub-second counter.
  - true: Enable RTC sub-second counter.
  - false: Disable RTC sub-second counter.

static inline uint32_t RTC_GetSubsecValue(const RTC_Type *base)

A read of 32KHZ sub-seconds counter.

Parameters

- base – RTC peripheral base address

Returns

Current value of the SUBSEC register

static inline void RTC_EnableWakeUpTimerInterruptFromDPD(RTC_Type *base, bool enable)

Enable the wake-up timer interrupt from deep power down mode.

Parameters

- base – RTC peripheral base address
- enable – Enable/Disable wake-up timer interrupt from deep power down mode.
  - true: Enable wake-up timer interrupt from deep power down mode.
  - false: Disable wake-up timer interrupt from deep power down mode.

static inline void RTC_EnableAlarmTimerInterruptFromDPD(RTC_Type *base, bool enable)

Enable the alarm timer interrupt from deep power down mode.

Parameters

- base – RTC peripheral base address
- enable – Enable/Disable alarm timer interrupt from deep power down mode.
  - true: Enable alarm timer interrupt from deep power down mode.
  - false: Disable alarm timer interrupt from deep power down mode.

static inline void RTC_EnableInterrupts(RTC_Type *base, uint32_t mask)

Enables the selected RTC interrupts.

Deprecated:

Do not use this function. It has been superceded by *RTC_EnableAlarmTimerInterruptFromDPD* and RTC_EnableWakeUpTimerInterruptFromDPD

Parameters

- base – RTC peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *rtc_interrupt_enable_t*

static inline void RTC_DisableInterrupts(RTC_Type *base, uint32_t mask)

Disables the selected RTC interrupts.

Deprecated:

Do not use this function. It has been superceded by *RTC_EnableAlarmTimerInterruptFromDPD* and RTC_EnableWakeUpTimerInterruptFromDPD

Parameters

- base – RTC peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *rtc_interrupt_enable_t*

static inline uint32_t RTC_GetEnabledInterrupts(RTC_Type *base)

Get the enabled RTC interrupts.

Deprecated:

Do not use this function. It will be deleted in next release version.

Parameters

- base – RTC peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *rtc_interrupt_enable_t*

static inline uint32_t RTC_GetStatusFlags(RTC_Type *base)

Get the RTC status flags.

Parameters

- base – RTC peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration *rtc_status_flags_t*

static inline void RTC_ClearStatusFlags(RTC_Type *base, uint32_t mask)

Clear the RTC status flags.

Parameters

- base – RTC peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration *rtc_status_flags_t*

static inline void RTC_EnableTimer(RTC_Type *base, bool enable)

Enable the RTC timer counter.

After calling this function, the RTC inner counter increments once a second when only using the RTC seconds timer (1hz), while the RTC inner wake-up timer countdown once a millisecond when using RTC wake-up timer (1KHZ) at the same time. RTC timer contain two timers, one is the RTC normal seconds timer, the other one is the RTC wake-up timer, the RTC enable bit is the master switch for the whole RTC timer, so user can use the RTC seconds (1HZ) timer independly, but they can't use the RTC wake-up timer (1KHZ) independently.

Parameters

- base – RTC peripheral base address
- enable – Enable/Disable RTC Timer counter.
  - true: Enable RTC Timer counter.
  - false: Disable RTC Timer counter.

static inline void RTC_StartTimer(RTC_Type *base)

Starts the RTC time counter.

Deprecated:

Do not use this function. It has been superceded by *RTC_EnableTimer*

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

Parameters

- base – RTC peripheral base address

static inline void RTC_StopTimer(RTC_Type *base)

Stops the RTC time counter.

Deprecated:

Do not use this function. It has been superceded by *RTC_EnableTimer*

RTC's seconds register can be written to only when the timer is stopped.

Parameters

- base – RTC peripheral base address

FSL_RTC_DRIVER_VERSION

Version 2.2.0

enum _rtc_interrupt_enable

List of RTC interrupts.

Values:

enumerator kRTC_AlarmInterruptEnable

Alarm interrupt.

enumerator kRTC_WakeupInterruptEnable

Wake-up interrupt.

enum _rtc_status_flags

List of RTC flags.

Values:

enumerator kRTC_AlarmFlag

Alarm flag

enumerator kRTC_WakeupFlag

1kHz wake-up timer flag

typedef enum *_rtc_interrupt_enable* rtc_interrupt_enable_t

List of RTC interrupts.

typedef enum *_rtc_status_flags* rtc_status_flags_t

List of RTC flags.

typedef struct *_rtc_datetime* rtc_datetime_t

Structure is used to hold the date and time.

static inline void RTC_SetSecondsTimerMatch(RTC_Type *base, uint32_t matchValue)

Set the RTC seconds timer (1HZ) MATCH value.

Parameters

- base – RTC peripheral base address
- matchValue – The value to be set into the RTC MATCH register

static inline uint32_t RTC_GetSecondsTimerMatch(RTC_Type *base)

Read actual RTC seconds timer (1HZ) MATCH value.

Parameters

- base – RTC peripheral base address

Returns

The actual RTC seconds timer (1HZ) MATCH value.

static inline void RTC_SetSecondsTimerCount(RTC_Type *base, uint32_t countValue)

Set the RTC seconds timer (1HZ) COUNT value.

Parameters

- base – RTC peripheral base address
- countValue – The value to be loaded into the RTC COUNT register

static inline uint32_t RTC_GetSecondsTimerCount(RTC_Type *base)

Read the actual RTC seconds timer (1HZ) COUNT value.

Parameters

- base – RTC peripheral base address

Returns

The actual RTC seconds timer (1HZ) COUNT value.

static inline void RTC_SetWakeupCount(RTC_Type *base, uint16_t wakeupValue)

Enable the RTC wake-up timer (1KHZ) and set countdown value to the RTC WAKE register.

Parameters

- base – RTC peripheral base address
- wakeupValue – The value to be loaded into the WAKE register in RTC wake-up timer (1KHZ).

static inline uint16_t RTC_GetWakeupCount(RTC_Type *base)

Read the actual value from the WAKE register value in RTC wake-up timer (1KHZ)

Read the WAKE register twice and compare the result, if the value match,the time can be used.

Parameters

- base – RTC peripheral base address

Returns

The actual value of the WAKE register value in RTC wake-up timer (1KHZ).

static inline void RTC_Reset(RTC_Type *base)

Perform a software reset on the RTC module.

This resets all RTC registers to their reset value. The bit is cleared by software explicitly clearing it.

Parameters

- base – RTC peripheral base address

struct __rtc__datetime

#include <fsl_rtc.h> Structure is used to hold the date and time.

Public Members

uint16_t year

Range from 1970 to 2099.

uint8_t month

Range from 1 to 12.

uint8_t day

Range from 1 to 31 (depending on month).

uint8_t hour

Range from 0 to 23.

uint8_t minute

Range from 0 to 59.

uint8_t second

Range from 0 to 59.

2.58 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);

2.59 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.

2.60 SPI: Serial Peripheral Interface Driver

2.61 SPI DMA Driver

status_t SPI_MasterTransferCreateHandleDMA(SPI_Type *base, *spi_dma_handle_t* *handle, *spi_dma_callback_t* callback, void *userData, *dma_handle_t* *txHandle, *dma_handle_t* *rxHandle)

Initialize the SPI master DMA handle.

This function initializes the SPI master DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

- base – SPI peripheral base address.
- handle – SPI handle pointer.
- callback – User callback function called at the end of a transfer.
- userData – User data for callback.
- txHandle – DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
- rxHandle – DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

status_t SPI_MasterTransferDMA(SPI_Type *base, *spi_dma_handle_t* *handle, *spi_transfer_t* *xfer)

Perform a non-blocking SPI transfer using DMA.

Note: This interface returned immediately after transfer initiates, users should call SPI_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

- base – SPI peripheral base address.
- handle – SPI DMA handle pointer.

- xfer – Pointer to dma transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_SPI_Busy – SPI is not idle, is running another transfer.

status_t SPI_MasterHalfDuplexTransferDMA(SPI_Type *base, *spi_dma_handle_t* *handle, *spi_half_duplex_transfer_t* *xfer)

Transfers a block of data using a DMA method.

This function using polling way to do the first half transimission and using DMA way to do the srcond half transimission, the transfer mechanism is half-duplex. When do the second half transimission, code will return right away. When all data is transferred, the callback function is called.

Parameters

- base – SPI base pointer
- handle – A pointer to the spi_master_dma_handle_t structure which stores the transfer state.
- xfer – A pointer to the spi_half_duplex_transfer_t structure.

Returns

status of status_t.

static inline *status_t* SPI_SlaveTransferCreateHandleDMA(SPI_Type *base, *spi_dma_handle_t* *handle, *spi_dma_callback_t* callback, void *userData, *dma_handle_t* *txHandle, *dma_handle_t* *rxHandle)

Initialize the SPI slave DMA handle.

This function initializes the SPI slave DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

- base – SPI peripheral base address.
- handle – SPI handle pointer.
- callback – User callback function called at the end of a transfer.
- userData – User data for callback.
- txHandle – DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
- rxHandle – DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

static inline *status_t* SPI_SlaveTransferDMA(SPI_Type *base, *spi_dma_handle_t* *handle, *spi_transfer_t* *xfer)

Perform a non-blocking SPI transfer using DMA.

Note: This interface returned immediately after transfer initiates, users should call SPI_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

- base – SPI peripheral base address.
- handle – SPI DMA handle pointer.
- xfer – Pointer to dma transfer structure.

Return values

- kStatus_Success – Successfully start a transfer.
- kStatus_InvalidArgument – Input argument is invalid.
- kStatus_SPI_Busy – SPI is not idle, is running another transfer.

void SPI_MasterTransferAbortDMA(SPI_Type *base, *spi_dma_handle_t* *handle)

Abort a SPI transfer using DMA.

Parameters

- base – SPI peripheral base address.
- handle – SPI DMA handle pointer.

status_t SPI_MasterTransferGetCountDMA(SPI_Type *base, *spi_dma_handle_t* *handle, size_t *count)

Gets the master DMA transfered bytes.

This function gets the master DMA transfered bytes.

Parameters

- base – SPI peripheral base address.
- handle – A pointer to the spi_dma_handle_t structure which stores the transfer state.
- count – A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

static inline void SPI_SlaveTransferAbortDMA(SPI_Type *base, *spi_dma_handle_t* *handle)

Abort a SPI transfer using DMA.

Parameters

- base – SPI peripheral base address.
- handle – SPI DMA handle pointer.

static inline *status_t* SPI_SlaveTransferGetCountDMA(SPI_Type *base, *spi_dma_handle_t* *handle, size_t *count)

Gets the slave DMA transfered bytes.

This function gets the slave DMA transfered bytes.

Parameters

- base – SPI peripheral base address.
- handle – A pointer to the spi_dma_handle_t structure which stores the transfer state.
- count – A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

FSL_SPI_DMA_DRIVER_VERSION

SPI DMA driver version.

typedef struct *_spi_dma_handle* spi_dma_handle_t

typedef void (*spi_dma_callback_t)(SPI_Type *base, *spi_dma_handle_t* *handle, *status_t* status, void *userData)

SPI DMA callback called at the end of transfer.

struct _spi_dma_handle

#include <fsl_spi_dma.h> SPI DMA transfer handle, users should not touch the content of the handle.

Public Members

SPI_Type *base

SPI base address

volatile bool txInProgress

Send transfer finished

volatile bool rxInProgress

Receive transfer finished

uint8_t bytesPerFrame

Bytes in a frame for SPI transfer

uint8_t lastwordBytes

The Bytes of lastword for master

uint16_t txDummy

The dummy data for TX.

uint32_t lastword

The last word for master TX.

dma_handle_t *txHandle

DMA handler for SPI send

dma_handle_t *rxHandle

DMA handler for SPI receive

spi_dma_callback_t callback

Callback for SPI DMA transfer

void *userData

User Data for SPI DMA callback

uint32_t state

Internal state of SPI DMA transfer

size_t transferSize
Bytes need to be transfer

uint32_t instance
Index of SPI instance

const uint8_t *txNextData
The pointer of next time tx data

size_t txRemainingBytes
lastwordBytes + txRemainingBytes is number of data to be send [in bytes]

uint8_t *rxNextData
The pointer of next time rx data

size_t rxRemainingBytes
Number of data to be received [in bytes]

bool isSlave
SPI work in slave mode.

2.62 SPI Driver

FSL_SPI_DRIVER_VERSION
SPI driver version.

enum _spi_xfer_option
SPI transfer option.
Values:

enumerator kSPI_FrameDelay
A delay may be inserted, defined in the DLY register.

enumerator kSPI_FrameAssert
SSEL will be deasserted at the end of a transfer.

enumerator kSPI_FrameAssertEachFrame
SSEL will be deasserted at the end of each frame.

enum _spi_shift_direction
SPI data shifter direction options.
Values:

enumerator kSPI_MsbFirst
Data transfers start with most significant bit.

enumerator kSPI_LsbFirst
Data transfers start with least significant bit.

enum _spi_clock_polarity
SPI clock polarity configuration.
Values:

enumerator kSPI_ClockPolarityActiveHigh
Active-high SPI clock (idles low).

enumerator kSPI_ClockPolarityActiveLow
Active-low SPI clock (idles high).

enum _spi_clock_phase
SPI clock phase configuration.
Values:

enumerator kSPI_ClockPhaseFirstEdge
First edge on SCK occurs at the middle of the first cycle of a data transfer.

enumerator kSPI_ClockPhaseSecondEdge
First edge on SCK occurs at the start of the first cycle of a data transfer.

enum _spi_txfifo_watermark
txFIFO watermark values
Values:

enumerator kSPI_TxFifo0
SPI tx watermark is empty

enumerator kSPI_TxFifo1
SPI tx watermark at 1 item

enumerator kSPI_TxFifo2
SPI tx watermark at 2 items

enumerator kSPI_TxFifo3
SPI tx watermark at 3 items

enumerator kSPI_TxFifo4
SPI tx watermark at 4 items

enumerator kSPI_TxFifo5
SPI tx watermark at 5 items

enumerator kSPI_TxFifo6
SPI tx watermark at 6 items

enumerator kSPI_TxFifo7
SPI tx watermark at 7 items

enum _spi_rxfifo_watermark
rxFIFO watermark values
Values:

enumerator kSPI_RxFifo1
SPI rx watermark at 1 item

enumerator kSPI_RxFifo2
SPI rx watermark at 2 items

enumerator kSPI_RxFifo3
SPI rx watermark at 3 items

enumerator kSPI_RxFifo4
SPI rx watermark at 4 items

enumerator kSPI_RxFifo5
SPI rx watermark at 5 items

enumerator kSPI_RxFifo6
SPI rx watermark at 6 items

enumerator kSPI_RxFifo7
SPI rx watermark at 7 items

enumerator kSPI_RxFifo8
SPI rx watermark at 8 items

enum _spi_data_width
Transfer data width.
Values:

enumerator kSPI_Data4Bits
4 bits data width

enumerator kSPI_Data5Bits
5 bits data width

enumerator kSPI_Data6Bits
6 bits data width

enumerator kSPI_Data7Bits
7 bits data width

enumerator kSPI_Data8Bits
8 bits data width

enumerator kSPI_Data9Bits
9 bits data width

enumerator kSPI_Data10Bits
10 bits data width

enumerator kSPI_Data11Bits
11 bits data width

enumerator kSPI_Data12Bits
12 bits data width

enumerator kSPI_Data13Bits
13 bits data width

enumerator kSPI_Data14Bits
14 bits data width

enumerator kSPI_Data15Bits
15 bits data width

enumerator kSPI_Data16Bits
16 bits data width

enum _spi_ssel
Slave select.
Values:

enumerator kSPI_Ssel0
Slave select 0

enumerator kSPI_Ssel1
Slave select 1

enumerator kSPI_Ssel2
Slave select 2

enumerator kSPI_Ssel3
Slave select 3

enum _spi_spol
ssel polarity
Values:

enumerator kSPI_Spol0ActiveHigh

enumerator kSPI_Spol1ActiveHigh

enumerator kSPI_Spol3ActiveHigh

enumerator kSPI_SpolActiveAllHigh

enumerator kSPI_SpolActiveAllLow

SPI transfer status.

Values:

enumerator kStatus_SPI_Busy
SPI bus is busy

enumerator kStatus_SPI_Idle
SPI is idle

enumerator kStatus_SPI_Error
SPI error

enumerator kStatus_SPI_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_SPI_Timeout
SPI timeout polling status flags.

enum _spi_interrupt_enable

SPI interrupt sources.

Values:

enumerator kSPI_RxLvlIrq
Rx level interrupt

enumerator kSPI_TxLvlIrq
Tx level interrupt

enum _spi_statusflags

SPI status flags.

Values:

enumerator kSPI_TxEmptyFlag
txFifo is empty

enumerator kSPI_TxNotFullFlag
txFifo is not full

enumerator kSPI_RxNotEmptyFlag
rxFIFO is not empty

enumerator kSPI_RxFullFlag
rxFIFO is full

typedef enum *_spi_xfer_option* spi_xfer_option_t
SPI transfer option.

typedef enum *_spi_shift_direction* spi_shift_direction_t
SPI data shifter direction options.

typedef enum *_spi_clock_polarity* spi_clock_polarity_t
SPI clock polarity configuration.

typedef enum *_spi_clock_phase* spi_clock_phase_t
SPI clock phase configuration.

typedef enum *_spi_txfifo_watermark* spi_txfifo_watermark_t
txFIFO watermark values

typedef enum *_spi_rxfifo_watermark* spi_rxfifo_watermark_t
rxFIFO watermark values

typedef enum *_spi_data_width* spi_data_width_t
Transfer data width.

typedef enum *_spi_ssel* spi_ssel_t
Slave select.

typedef enum *_spi_spol* spi_spol_t
ssel polarity

typedef struct *_spi_delay_config* spi_delay_config_t
SPI delay time configure structure. Note: The DLY register controls several programmable delays related to SPI signalling, it stands for how many SPI clock time will be inserted. The maxinun value of these delay time is 15.

typedef struct *_spi_master_config* spi_master_config_t
SPI master user configure structure.

typedef struct *_spi_slave_config* spi_slave_config_t
SPI slave user configure structure.

typedef struct *_spi_transfer* spi_transfer_t
SPI transfer structure.

typedef struct *_spi_half_duplex_transfer* spi_half_duplex_transfer_t
SPI half-duplex(master only) transfer structure.

typedef struct *_spi_config* spi_config_t

Internal configuration structure used in 'spi' and 'spi_dma' driver.

typedef struct *_spi_master_handle* spi_master_handle_t

Master handle type.

typedef *spi_master_handle_t* spi_slave_handle_t

Slave handle type.

typedef void (*spi_master_callback_t)(SPI_Type *base, *spi_master_handle_t* *handle, *status_t* status, void *userData)

SPI master callback for finished transmit.

typedef void (*spi_slave_callback_t)(SPI_Type *base, *spi_slave_handle_t* *handle, *status_t* status, void *userData)

SPI slave callback for finished transmit.

typedef void (*flexcomm_spi_master_irq_handler_t)(SPI_Type *base, *spi_master_handle_t* *handle)

Typedef for master interrupt handler.

typedef void (*flexcomm_spi_slave_irq_handler_t)(SPI_Type *base, *spi_slave_handle_t* *handle)

Typedef for slave interrupt handler.

volatile uint8_t s_dummyData[]

SPI default SSEL COUNT.

Global variable for dummy data value setting.

SPI_DUMMYDATA

SPI dummy transfer data, the data is sent while txBuff is NULL.

SPI_RETRY_TIMES

Retry times for waiting flag.

SPI_DATA(n)

SPI_CTRLMASK

SPI_ASSERTNUM_SSEL(n)

SPI_DEASSERTNUM_SSEL(n)

SPI_DEASSERT_ALL

SPI_FIFOWR_FLAGS_MASK

SPI_FIFOTRIG_TXLVL_GET(base)

SPI_FIFOTRIG_RXLVL_GET(base)

struct _spi_delay_config

#include <fsl_spi.h> SPI delay time configure structure. Note: The DLY register controls several programmable delays related to SPI signalling, it stands for how many SPI clock time will be inserted. The maxinun value of these delay time is 15.

Public Members

uint8_t preDelay

Delay between SSEL assertion and the beginning of transfer.

uint8_t postDelay

Delay between the end of transfer and SSEL deassertion.

uint8_t frameDelay

Delay between frame to frame.

uint8_t transferDelay

Delay between transfer to transfer.

struct _spi_master_config

#include <fsl_spi.h> SPI master user configure structure.

Public Members

bool enableLoopback

Enable loopback for test purpose

bool enableMaster

Enable SPI at initialization time

spi_clock_polarity_t polarity

Clock polarity

spi_clock_phase_t phase

Clock phase

spi_shift_direction_t direction

MSB or LSB

uint32_t baudRate_Bps

Baud Rate for SPI in Hz

spi_data_width_t dataWidth

Width of the data

spi_ssel_t sselNum

Slave select number

spi_spol_t sselPol
Configure active CS polarity

uint8_t txWatermark
txFIFO watermark

uint8_t rxWatermark
rxFIFO watermark

spi_delay_config_t delayConfig
Delay configuration.

struct __spi__slave__config
#include <fsl_spi.h> SPI slave user configure structure.

Public Members

bool enableSlave
Enable SPI at initialization time

spi_clock_polarity_t polarity
Clock polarity

spi_clock_phase_t phase
Clock phase

spi_shift_direction_t direction
MSB or LSB

spi_data_width_t dataWidth
Width of the data

spi_spol_t sselPol
Configure active CS polarity

uint8_t txWatermark
txFIFO watermark

uint8_t rxWatermark
rxFIFO watermark

struct __spi__transfer
#include <fsl_spi.h> SPI transfer structure.

Public Members

const uint8_t *txData
: Send buffer

uint8_t *rxData
: Receive buffer

uint32_t configFlags
: Additional option to control transfer, *spi_xfer_option_t*.

size_t dataSize
: Transfer bytes

struct __spi__half__duplex__transfer
: *#include <fsl_spi.h>* SPI half-duplex(master only) transfer structure.

Public Members

const uint8_t *txData
: Send buffer

uint8_t *rxData
: Receive buffer

size_t txDataSize
: Transfer bytes for transmit

size_t rxDataSize
: Transfer bytes

uint32_t configFlags
: Transfer configuration flags, *spi_xfer_option_t*.

bool isPcsAssertInTransfer
: If PCS pin keep assert between transmit and receive. true for assert and false for de-assert.

bool isTransmitFirst
: True for transmit first and false for receive first.

struct __spi__config
: *#include <fsl_spi.h>* Internal configuration structure used in 'spi' and 'spi_dma' driver.

struct __spi__master__handle
: *#include <fsl_spi.h>* SPI transfer handle structure.

Public Members

const uint8_t *volatile txData
: Transfer buffer

uint8_t *volatile rxData
: Receive buffer

volatile size_t txRemainingBytes
: Number of data to be transmitted [in bytes]

volatile size_t rxRemainingBytes
: Number of data to be received [in bytes]

volatile int8_t toReceiveCount
: The number of data expected to receive in data width. Since the received count and sent count should be the same to complete the transfer, if the sent count is x and the received count is y, toReceiveCount is x-y.

size_t totalByteCount
: A number of transfer bytes

volatile uint32_t state
: SPI internal state

spi_master_callback_t callback
: SPI callback

void *userData
: Callback parameter

uint8_t dataWidth
: Width of the data [Valid values: 1 to 16]

uint8_t txWatermark
: txFIFO watermark

uint8_t rxWatermark
: rxFIFO watermark

uint32_t txControl
: Precomputed FIFOWR control bits for non-last frames in interrupt transfer.

uint32_t lastControl
: Precomputed FIFOWR control bits for the last frame in interrupt transfer.

2.63 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.

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 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.

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.

2.64 USART: Universal Synchronous/Asynchronous Receiver/Transmitter Driver

2.65 USART DMA Driver

status_t USART_TransferCreateHandleDMA(USART_Type *base, *usart_dma_handle_t* *handle, *usart_dma_transfer_callback_t* callback, void *userData, *dma_handle_t* *txDmaHandle, *dma_handle_t* *rxDmaHandle)

Initializes the USART handle which is used in transactional functions.

Parameters

- base – USART peripheral base address.
- handle – Pointer to usart_dma_handle_t structure.
- callback – Callback function.
- userData – User data.
- txDmaHandle – User-requested DMA handle for TX DMA transfer.
- rxDmaHandle – User-requested DMA handle for RX DMA transfer.

status_t USART_TransferSendDMA(USART_Type *base, *usart_dma_handle_t* *handle, *usart_transfer_t* *xfer)

Sends data using DMA.

This function sends data using DMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.
- xfer – USART DMA transfer structure. See *usart_transfer_t*.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_USART_TxBusy – Previous transfer on going.
- kStatus_InvalidArgument – Invalid argument.

status_t USART_TransferReceiveDMA(USART_Type *base, *usart_dma_handle_t* *handle, *usart_transfer_t* *xfer)

Receives data using DMA.

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – USART peripheral base address.
- handle – Pointer to usart_dma_handle_t structure.
- xfer – USART DMA transfer structure. See *usart_transfer_t*.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_USART_RxBusy – Previous transfer on going.
- kStatus_InvalidArgument – Invalid argument.

void USART_TransferAbortSendDMA(USART_Type *base, *usart_dma_handle_t* *handle)

Aborts the sent data using DMA.

This function aborts send data using DMA.

Parameters

- base – USART peripheral base address
- handle – Pointer to usart_dma_handle_t structure

void USART_TransferAbortReceiveDMA(USART_Type *base, *usart_dma_handle_t* *handle)

Aborts the received data using DMA.

This function aborts the received data using DMA.

Parameters

- base – USART peripheral base address
- handle – Pointer to usart_dma_handle_t structure

status_t USART_TransferGetReceiveCountDMA(USART_Type *base, *usart_dma_handle_t* *handle, uint32_t *count)

Get the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – USART peripheral base address.
- handle – USART 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;

status_t USART_TransferGetSendCountDMA(USART_Type *base, *usart_dma_handle_t* *handle, uint32_t *count)

Get the number of bytes that have been sent.

This function gets the number of bytes that have been sent.

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.
- count – Sent bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

FSL_USART_DMA_DRIVER_VERSION

USART dma driver version.

typedef struct *_usart_dma_handle* usart_dma_handle_t

typedef void (*usart_dma_transfer_callback_t)(USART_Type *base, *usart_dma_handle_t* *handle, *status_t* status, void *userData)

UART transfer callback function.

struct _usart_dma_handle

#include <fsl_usart_dma.h> UART DMA handle.

Public Members

USART_Type *base

UART peripheral base address.

usart_dma_transfer_callback_t callback

Callback function.

void *userData

UART callback function parameter.

size_t rxDataSizeAll

Size of the data to receive.

size_t txDataSizeAll

Size of the data to send out.

dma_handle_t *txDmaHandle

The DMA TX channel used.

dma_handle_t *rxDmaHandle

The DMA RX channel used.

volatile uint8_t txState

TX transfer state.

volatile uint8_t rxState

RX transfer state

2.66 USART Driver

status_t USART_Init(USART_Type *base, const *usart_config_t* *config, uint32_t srcClock_Hz)

Initializes a USART instance with user configuration structure and peripheral clock.

This function configures the USART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the *USART_GetDefaultConfig()* function. Example below shows how to use this API to configure USART.

```
usart_config_t usartConfig;
usartConfig.baudRate_Bps = 115200U;
usartConfig.parityMode = kUSART_ParityDisabled;
usartConfig.stopBitCount = kUSART_OneStopBit;
USART_Init(USART1, &usartConfig, 20000000U);
```

Parameters

- base – USART peripheral base address.
- config – Pointer to user-defined configuration structure.
- srcClock_Hz – USART clock source frequency in HZ.

Return values

- kStatus_USART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_InvalidArgument – USART base address is not valid
- kStatus_Success – Status USART initialize succeed

void USART_Deinit(USART_Type *base)

Deinitializes a USART instance.

This function waits for TX complete, disables TX and RX, and disables the USART clock.

Parameters

- base – USART peripheral base address.

void USART_GetDefaultConfig(*usart_config_t* *config)

Gets the default configuration structure.

This function initializes the USART configuration structure to a default value. The default values are: usartConfig->baudRate_Bps = 115200U; usartConfig->parityMode = kUSART_ParityDisabled; usartConfig->stopBitCount = kUSART_OneStopBit; usartConfig->bitCountPerChar = kUSART_8BitsPerChar; usartConfig->loopback = false; usartConfig->enableTx = false; usartConfig->enableRx = false;

Parameters

- config – Pointer to configuration structure.

status_t USART_SetBaudRate(USART_Type *base, uint32_t baudrate_Bps, uint32_t srcClock_Hz)

Sets the USART instance baud rate.

This function configures the USART module baud rate. This function is used to update the USART module baud rate after the USART module is initialized by the USART_Init.

```
USART_SetBaudRate(USART1, 115200U, 20000000U);
```

Parameters

- base – USART peripheral base address.
- baudrate_Bps – USART baudrate to be set.
- srcClock_Hz – USART clock source frequency in HZ.

Return values

- kStatus_USART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – Set baudrate succeed.
- kStatus_InvalidArgument – One or more arguments are invalid.

status_t USART_Enable32kMode(USART_Type *base, uint32_t baudRate_Bps, bool enableMode32k, uint32_t srcClock_Hz)

Enable 32 kHz mode which USART uses clock from the RTC oscillator as the clock source.

Please note that in order to use a 32 kHz clock to operate USART properly, the RTC oscillator and its 32 kHz output must be manully enabled by user, by calling RTC_Init and setting SYSCON_RTCOSCCTRL_EN bit to 1. And in 32kHz clocking mode the USART can only work at 9600 baudrate or at the baudrate that 9600 can evenly divide, eg: 4800, 3200.

Parameters

- base – USART peripheral base address.
- baudRate_Bps – USART baudrate to be set..
- enableMode32k – true is 32k mode, false is normal mode.
- srcClock_Hz – USART clock source frequency in HZ.

Return values

- kStatus_USART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – Set baudrate succeed.
- kStatus_InvalidArgument – One or more arguments are invalid.

void USART_Enable9bitMode(USART_Type *base, bool enable)

Enable 9-bit data mode for USART.

This function set the 9-bit mode for USART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- base – USART peripheral base address.
- enable – true to enable, false to disable.

static inline void USART_SetMatchAddress(USART_Type *base, uint8_t address)

Set the USART slave address.

This function configures the address for USART module that works as slave in 9-bit data mode. When the address detection is enabled, 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 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 USART 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 – USART peripheral base address.
- address – USART slave address.

static inline void USART_EnableMatchAddress(USART_Type *base, bool match)

Enable the USART match address feature.

Parameters

- base – USART peripheral base address.
- match – true to enable match address, false to disable.

static inline uint32_t USART_GetStatusFlags(USART_Type *base)

Get USART status flags.

This function get all USART status flags, the flags are returned as the logical OR value of the enumerators *_usart_flags*. To check a specific status, compare the return value with enumerators in *_usart_flags*. For example, to check whether the TX is empty:

```
if (kUSART_TxFifoNotFullFlag & USART_GetStatusFlags(USART1))
{
    ...
}
```

Parameters

- base – USART peripheral base address.

Returns

USART status flags which are ORed by the enumerators in the _usart_flags.

static inline void USART_ClearStatusFlags(USART_Type *base, uint32_t mask)

Clear USART status flags.

This function clear supported USART status flags. The mask is a logical OR of enumeration members. See *kUSART_AllClearFlags*. For example:

```
USART_ClearStatusFlags(USART1, kUSART_TxError | kUSART_RxError)
```

Parameters

- base – USART peripheral base address.
- mask – status flags to be cleared.

static inline void USART_EnableInterrupts(USART_Type *base, uint32_t mask)

Enables USART interrupts according to the provided mask.

This function enables the USART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See *_usart_interrupt_enable*. For example, to enable TX empty interrupt and RX full interrupt:

```
USART_EnableInterrupts(USART1, kUSART_TxLevelInterruptEnable | kUSART_
↪RxLevelInterruptEnable);
```

Parameters

- base – USART peripheral base address.
- mask – The interrupts to enable. Logical OR of *_usart_interrupt_enable*.

static inline void USART_DisableInterrupts(USART_Type *base, uint32_t mask)

Disables USART interrupts according to a provided mask.

This function disables the USART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See *_usart_interrupt_enable*. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
USART_DisableInterrupts(USART1, kUSART_TxLevelInterruptEnable | kUSART_
↪RxLevelInterruptEnable);
```

Parameters

- base – USART peripheral base address.
- mask – The interrupts to disable. Logical OR of *_usart_interrupt_enable*.

static inline uint32_t USART_GetEnabledInterrupts(USART_Type *base)

Returns enabled USART interrupts.

This function returns the enabled USART interrupts.

Parameters

- base – USART peripheral base address.

static inline void USART_EnableTxDMA(USART_Type *base, bool enable)

Enable DMA for Tx.

static inline void USART_EnableRxDMA(USART_Type *base, bool enable)

Enable DMA for Rx.

static inline void USART_EnableCTS(USART_Type *base, bool enable)

Enable CTS. This function will determine whether CTS is used for flow control.

Parameters

- base – USART peripheral base address.
- enable – Enable CTS or not, true for enable and false for disable.

static inline void USART_EnableContinuousSCLK(USART_Type *base, bool enable)

Continuous Clock generation. By default, SCLK is only output while data is being transmitted in synchronous mode. Enable this funciton, SCLK will run continuously in synchronous mode, allowing characters to be received on Un_RxD independently from transmission on Un_TXD).

Parameters

- base – USART peripheral base address.
- enable – Enable Continuous Clock generation mode or not, true for enable and false for disable.

static inline void USART_EnableAutoClearSCLK(USART_Type *base, bool enable)

Enable Continuous Clock generation bit auto clear. While enable this cuntion, the Continuous Clock bit is automatically cleared when a complete character has been received. This bit is cleared at the same time.

Parameters

- base – USART peripheral base address.

- enable – Enable auto clear or not, true for enable and false for disable.

static inline void USART_SetRxFifoWatermark(USART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – USART peripheral base address.
- water – Rx FIFO watermark.

static inline void USART_SetTxFifoWatermark(USART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – USART peripheral base address.
- water – Tx FIFO watermark.

static inline void USART_WriteByte(USART_Type *base, uint8_t data)

Writes to the FIFOWR register.

This function writes data to the txFIFO directly. The upper layer must ensure that txFIFO has space for data to write before calling this function.

Parameters

- base – USART peripheral base address.
- data – The byte to write.

static inline uint8_t USART_ReadByte(USART_Type *base)

Reads the FIFORD register directly.

This function reads data from the rxFIFO directly. The upper layer must ensure that the rxFIFO is not empty before calling this function.

Parameters

- base – USART peripheral base address.

Returns

The byte read from USART data register.

static inline uint8_t USART_GetRxFifoCount(USART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – USART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t USART_GetTxFifoCount(USART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – USART peripheral base address.

Returns

tx FIFO data count.

void USART_SendAddress(USART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

Parameters

- base – USART peripheral base address.

- address – USART slave address.

status_t USART_WriteBlocking(USART_Type *base, const uint8_t *data, size_t length)

Writes to the TX register using a blocking method.

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

Parameters

- base – USART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_USART_Timeout – Transmission timed out and was aborted.
- kStatus_InvalidArgument – Invalid argument.
- kStatus_Success – Successfully wrote all data.

status_t USART_ReadBlocking(USART_Type *base, uint8_t *data, size_t length)

Read RX data register using a blocking method.

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data and read data from the TX register.

Parameters

- base – USART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_USART_FramingError – Receiver overrun happened while receiving data.
- kStatus_USART_ParityError – Noise error happened while receiving data.
- kStatus_USART_NoiseError – Framing error happened while receiving data.
- kStatus_USART_RxError – Overflow or underflow rxFIFO happened.
- kStatus_USART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t USART_TransferCreateHandle(USART_Type *base, *usart_handle_t* *handle, *usart_transfer_callback_t* callback, void *userData)

Initializes the USART handle.

This function initializes the USART handle which can be used for other USART transactional APIs. Usually, for a specified USART instance, call this API once to get the initialized handle.

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

status_t USART_TransferSendNonBlocking(USART_Type *base, *usart_handle_t* *handle, *usart_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 the IRQ handler, the USART driver calls the callback function and passes the *kStatus_USART_TxIdle* as status parameter.

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.
- xfer – USART transfer structure. See *usart_transfer_t*.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_USART_TxBusy – Previous transmission still not finished, data not all written to TX register yet.
- kStatus_InvalidArgument – Invalid argument.

void USART_TransferStartRingBuffer(USART_Type *base, *usart_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 USART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the *USART_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 the 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 – USART peripheral base address.
- handle – USART handle pointer.
- ringBuffer – Start address of the ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

void USART_TransferStopRingBuffer(USART_Type *base, *usart_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 – USART peripheral base address.
- handle – USART handle pointer.

size_t USART_TransferGetRxRingBufferLength(*usart_handle_t* *handle)

Get the length of received data in RX ring buffer.

Parameters

- handle – USART handle pointer.

Returns
Length of received data in RX ring buffer.

void USART_TransferAbortSend(USART_Type *base, *usart_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 still not sent out.

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.

status_t USART_TransferGetSendCount(USART_Type *base, *usart_handle_t* *handle, uint32_t *count)

Get 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 interrupt method.

Parameters

- base – USART peripheral base address.
- handle – USART 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 USART_TransferReceiveNonBlocking(USART_Type *base, *usart_handle_t* *handle, *usart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using an interrupt method.

This function receives data using an 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 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 to read, the receive request is saved by the USART driver. When the new data arrives, the receive request is serviced first. When all data is received, the USART driver notifies the upper layer through a callback function and passes the status parameter *kStatus_USART_RxIdle*. For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the xfer->data and this function returns with the parameter receivedBytes set to 5. For the left 5 bytes, newly arrived data is saved from the xfer->data[5]. When 5 bytes are received, the USART 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 the xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.
- xfer – USART transfer structure, see *usart_transfer_t*.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into transmit queue.
- kStatus_USART_RxBusy – Previous receive request is not finished.

- kStatus_InvalidArgument – Invalid argument.

void USART_TransferAbortReceive(USART_Type *base, *usart_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 – USART peripheral base address.
- handle – USART handle pointer.

status_t USART_TransferGetReceiveCount(USART_Type *base, *usart_handle_t* *handle, uint32_t *count)

Get the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – USART peripheral base address.
- handle – USART 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 USART_TransferHandleIRQ(USART_Type *base, *usart_handle_t* *handle)

USART IRQ handle function.

This function handles the USART transmit and receive IRQ request.

Parameters

- base – USART peripheral base address.
- handle – USART handle pointer.

FSL_USART_DRIVER_VERSION

USART driver version.

Error codes for the USART driver.

Values:

enumerator kStatus_USART_TxBusy

Transmitter is busy.

enumerator kStatus_USART_RxBusy

Receiver is busy.

enumerator kStatus_USART_TxIdle

USART transmitter is idle.

enumerator kStatus_USART_RxIdle
USART receiver is idle.

enumerator kStatus_USART_TxError
Error happens on txFIFO.

enumerator kStatus_USART_RxError
Error happens on rxFIFO.

enumerator kStatus_USART_RxRingBufferOverrun
Error happens on rx ring buffer

enumerator kStatus_USART_NoiseError
USART noise error.

enumerator kStatus_USART_FramingError
USART framing error.

enumerator kStatus_USART_ParityError
USART parity error.

enumerator kStatus_USART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_USART_RxStart
Start is detected on the receiver input

enum _usart_sync_mode
USART synchronous mode.
Values:

enumerator kUSART_SyncModeDisabled
Asynchronous mode.

enumerator kUSART_SyncModeSlave
Synchronous slave mode.

enumerator kUSART_SyncModeMaster
Synchronous master mode.

enum _usart_parity_mode
USART parity mode.
Values:

enumerator kUSART_ParityDisabled
Parity disabled

enumerator kUSART_ParityEven
Parity enabled, type even, bit setting: PE|PT = 10

enumerator kUSART_ParityOdd
Parity enabled, type odd, bit setting: PE|PT = 11

enum _usart_stop_bit_count
USART stop bit count.
Values:

enumerator kUSART_OneStopBit
One stop bit

enumerator kUSART_TwoStopBit
Two stop bits

enum _usart_data_len
USART data size.
Values:

enumerator kUSART_7BitsPerChar
Seven bit mode

enumerator kUSART_8BitsPerChar
Eight bit mode

enum _usart_clock_polarity
USART clock polarity configuration, used in sync mode.
Values:

enumerator kUSART_RxSampleOnFallingEdge
Un_RXD is sampled on the falling edge of SCLK.

enumerator kUSART_RxSampleOnRisingEdge
Un_RXD is sampled on the rising edge of SCLK.

enum _usart_txfifo_watermark
txFIFO watermark values
Values:

enumerator kUSART_TxFifo0
USART tx watermark is empty

enumerator kUSART_TxFifo1
USART tx watermark at 1 item

enumerator kUSART_TxFifo2
USART tx watermark at 2 items

enumerator kUSART_TxFifo3
USART tx watermark at 3 items

enumerator kUSART_TxFifo4
USART tx watermark at 4 items

enumerator kUSART_TxFifo5
USART tx watermark at 5 items

enumerator kUSART_TxFifo6
USART tx watermark at 6 items

enumerator kUSART_TxFifo7
USART tx watermark at 7 items

enum _usart_rxfifo_watermark
rxFIFO watermark values
Values:

enumerator kUSART_RxFifo1
USART rx watermark at 1 item

enumerator kUSART_RxFifo2
USART rx watermark at 2 items

enumerator kUSART_RxFifo3
USART rx watermark at 3 items

enumerator kUSART_RxFifo4
USART rx watermark at 4 items

enumerator kUSART_RxFifo5
USART rx watermark at 5 items

enumerator kUSART_RxFifo6
USART rx watermark at 6 items

enumerator kUSART_RxFifo7
USART rx watermark at 7 items

enumerator kUSART_RxFifo8
USART rx watermark at 8 items

enum _usart_interrupt_enable
USART interrupt configuration structure, default settings all disabled.
Values:

enumerator kUSART_TxErrorInterruptEnable

enumerator kUSART_RxErrorInterruptEnable

enumerator kUSART_TxLevelInterruptEnable

enumerator kUSART_RxLevelInterruptEnable

enumerator kUSART_TxIdleInterruptEnable
Transmitter idle.

enumerator kUSART_CtsChangeInterruptEnable
Change in the state of the CTS input.

enumerator kUSART_RxBreakChangeInterruptEnable
Break condition asserted or deasserted.

enumerator kUSART_RxStartInterruptEnable
Rx start bit detected.

enumerator kUSART_FramingErrorInterruptEnable
Framing error detected.

enumerator kUSART_ParityErrorInterruptEnable
Parity error detected.

enumerator kUSART_NoiseErrorInterruptEnable
Noise error detected.

enumerator kUSART_AutoBaudErrorInterruptEnable
Auto baudrate error detected.

enumerator kUSART_AllInterruptEnables

enum _usart_flags
USART status flags.
This provides constants for the USART status flags for use in the USART functions.
Values:

enumerator kUSART_TxError
TXERR bit, sets if TX buffer is error

enumerator kUSART_RxError
RXERR bit, sets if RX buffer is error

enumerator kUSART_TxFifoEmptyFlag
TXEMPTY bit, sets if TX buffer is empty

enumerator kUSART_TxFifoNotFullFlag
TXNOTFULL bit, sets if TX buffer is not full

enumerator kUSART_RxFifoNotEmptyFlag
RXNOEMPTY bit, sets if RX buffer is not empty

enumerator kUSART_RxFifoFullFlag
RXFULL bit, sets if RX buffer is full

enumerator kUSART_RxIdleFlag
Receiver idle.

enumerator kUSART_TxIdleFlag
Transmitter idle.

enumerator kUSART_CtsAssertFlag
CTS signal high.

enumerator kUSART_CtsChangeFlag
CTS signal changed interrupt status.

enumerator kUSART_BreakDetectFlag
Break detected. Self cleared when rx pin goes high again.

enumerator kUSART_BreakDetectChangeFlag
Break detect change interrupt flag. A change in the state of receiver break detection.

enumerator kUSART_RxStartFlag
Rx start bit detected interrupt flag.

enumerator kUSART_FramingErrorFlag
Framing error interrupt flag.

enumerator kUSART_ParityErrorFlag
parity error interrupt flag.

enumerator kUSART_NoiseErrorFlag
Noise error interrupt flag.

enumerator kUSART_AutobaudErrorFlag
Auto baudrate error interrupt flag, caused by the baudrate counter timeout before the end of start bit.

enumerator kUSART_AllClearFlags

typedef enum *_usart_sync_mode* usart_sync_mode_t
USART synchronous mode.

typedef enum *_usart_parity_mode* usart_parity_mode_t
USART parity mode.

typedef enum *_usart_stop_bit_count* usart_stop_bit_count_t

USART stop bit count.

typedef enum *_usart_data_len* usart_data_len_t

USART data size.

typedef enum *_usart_clock_polarity* usart_clock_polarity_t

USART clock polarity configuration, used in sync mode.

typedef enum *_usart_txfifo_watermark* usart_txfifo_watermark_t

txFIFO watermark values

typedef enum *_usart_rxfifo_watermark* usart_rxfifo_watermark_t

rxFIFO watermark values

typedef struct *_usart_config* usart_config_t

USART configuration structure.

typedef struct *_usart_transfer* usart_transfer_t

USART transfer structure.

typedef struct *_usart_handle* usart_handle_t

typedef void (*usart_transfer_callback_t)(USART_Type *base, *usart_handle_t* *handle, *status_t* status, void *userData)

USART transfer callback function.

typedef void (*flexcomm_usart_irq_handler_t)(USART_Type *base, *usart_handle_t* *handle)

Typedef for usart interrupt handler.

uint32_t USART_GetInstance(USART_Type *base)

Returns instance number for USART peripheral base address.

USART_FIFOTRIG_TXLVL_GET(base)

USART_FIFOTRIG_RXLVL_GET(base)

UART_RETRY_TIMES

Retry times for waiting flag.

Defining to zero means to keep waiting for the flag until it is assert/deassert in blocking transfer, otherwise the program will wait until the UART_RETRY_TIMES counts down to 0, if the flag still remains unchanged then program will return kStatus_USART_Timeout. It is not advised to use this macro in formal application to prevent any hardware error because the actual wait period is affected by the compiler and optimization.

struct _usart_config

#include <fsl_usart.h> USART configuration structure.

Public Members

uint32_t baudRate_Bps
: USART baud rate

usart_parity_mode_t parityMode
: Parity mode, disabled (default), even, odd

usart_stop_bit_count_t stopBitCount
: Number of stop bits, 1 stop bit (default) or 2 stop bits

usart_data_len_t bitCountPerChar
: Data length - 7 bit, 8 bit

bool loopback
: Enable peripheral loopback

bool enableRx
: Enable RX

bool enableTx
: Enable TX

bool enableContinuousSCLK
: USART continuous Clock generation enable in synchronous master mode.

bool enableMode32k
: USART uses 32 kHz clock from the RTC oscillator as the clock source.

bool enableHardwareFlowControl
: Enable hardware control RTS/CTS

bool enable485ControlOutput
: Configure RTS signal to provide an output enable signal to control an RS-485 transceiver.

bool enableActiveHighRts
: Output enable polarity.

bool extendRtsOutput
: Output Enable Turnaround time enable for RS-485 operation.

usart_txfifo_watermark_t txWatermark
: txFIFO watermark

usart_rxfifo_watermark_t rxWatermark
: rxFIFO watermark

usart_sync_mode_t syncMode
: Transfer mode select - asynchronous, synchronous master, synchronous slave.

usart_clock_polarity_t clockPolarity
Selects the clock polarity and sampling edge in synchronous mode.

struct __usart__transfer
#include <fsl_usart.h> USART transfer structure.

Public Members

size_t dataSize
The byte count to be transfer.

struct __usart__handle
#include <fsl_usart.h> USART 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.

size_t txDataSizeAll
Size of the data to send out.

uint8_t *volatile rxData
Address of remaining data to receive.

volatile size_t rxDataSize
Size of the remaining data to receive.

size_t rxDataSizeAll
Size of the data to receive.

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.

usart_transfer_callback_t callback
Callback function.

void *userData
USART callback function parameter.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

uint8_t txWatermark
txFIFO watermark

uint8_t rxWatermark
rxFIFO watermark

union ___unnamed19___

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.

2.67 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.

2.68 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.

2.69 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;
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.

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 Motor Control

3.4.1 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.

3.5 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.

3.5.1 Quick links

- *Release Notes*

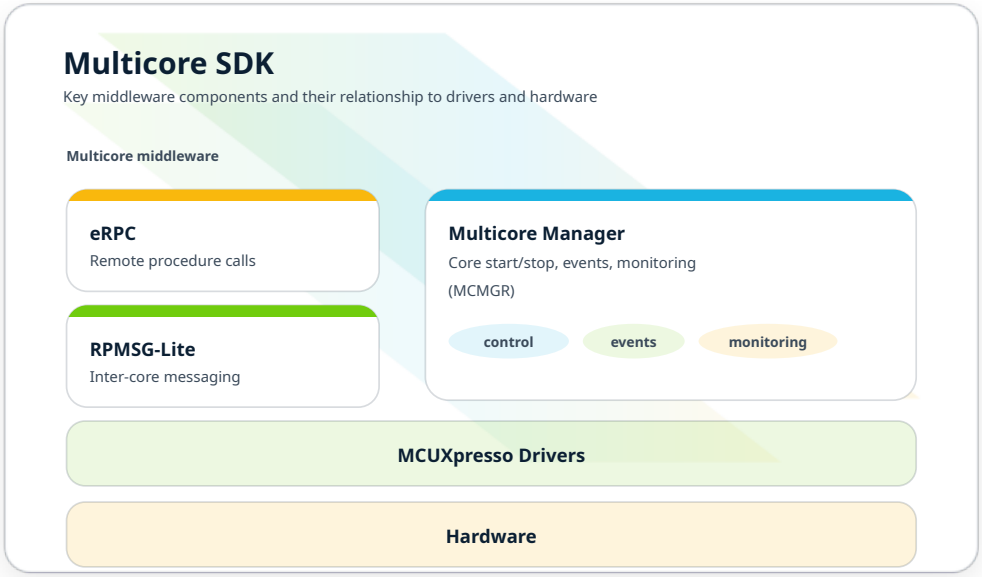


Fig. 1: Multicore SDK software stack (simplified).

- *Getting Started*
- *Multicore SDK Changelog*

3.5.2 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:

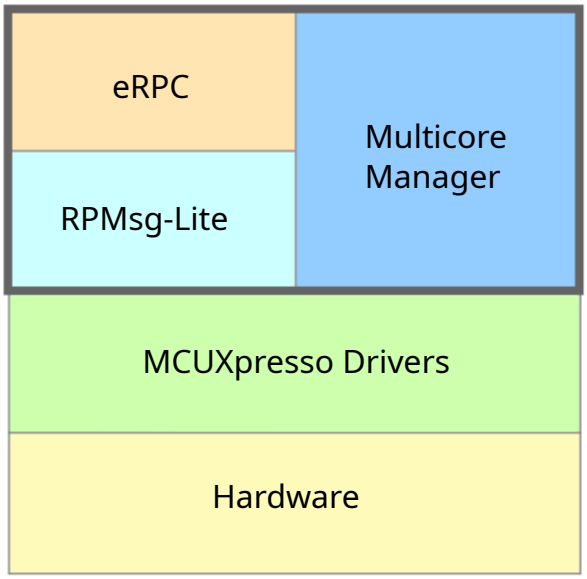
- Multicore: <MCUXpressoSDK_install_dir>/examples/multicore_examples/

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

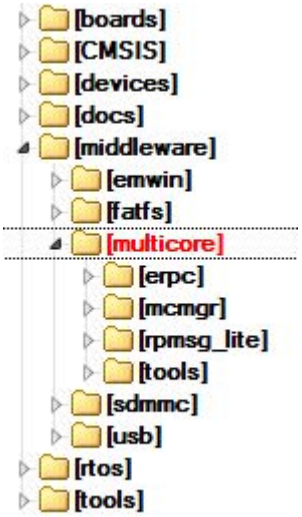
- MCSDK Release Notes
- *MCSDK Changelog*
- Component repositories (within MCSDK):
  - RPMSG-Lite
  - Multicore Manager (MCMGR)
  - eRPC
- eRPC detailed Getting Started guide:

– 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:

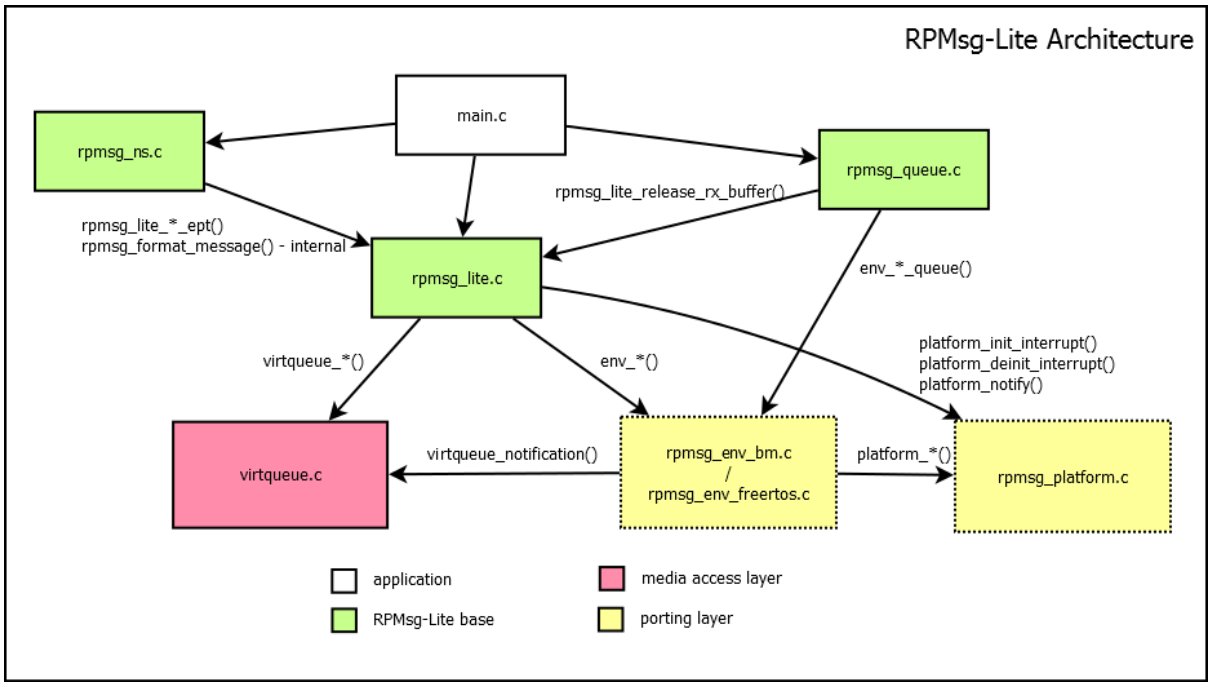
Component / Configuration	Flash [B]	RAM [B]
OpenAMP RPMsg / Release (reference)	5547	456 + dynamic
RPMsg-Lite / Dynamic API, Release	3462	56 + dynamic
Relative Difference [%]	~62.4%	~12.3%
RPMsg-Lite / Static API (no malloc), Release	2926	352
Relative Difference [%]	~52.7%	~77.2%

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).

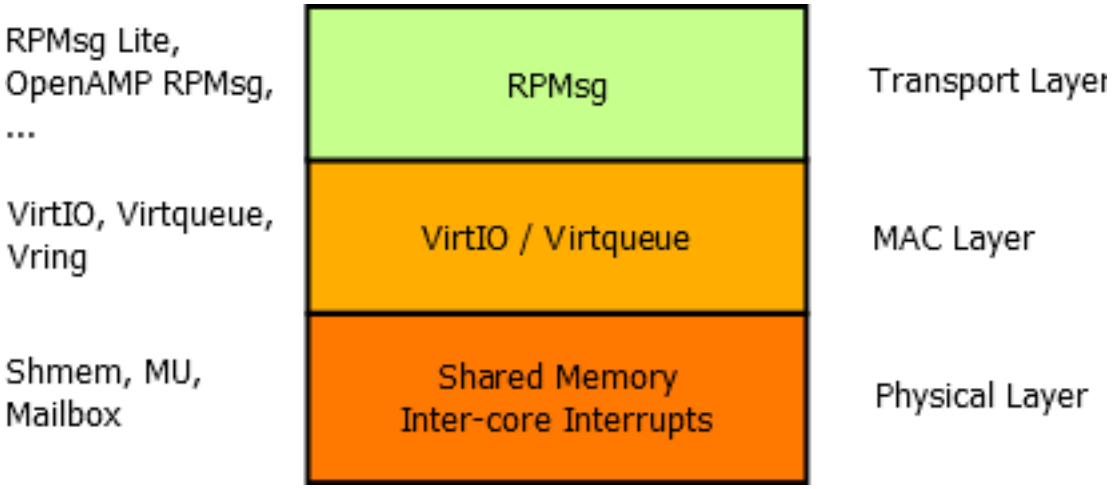
The actual “media access” layer is implemented in virtqueue.c, which is one of the few files shared with the OpenAMP implementation. This layer mainly defines the shared memory model, and internally defines used components such as vring or virtqueue.

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.

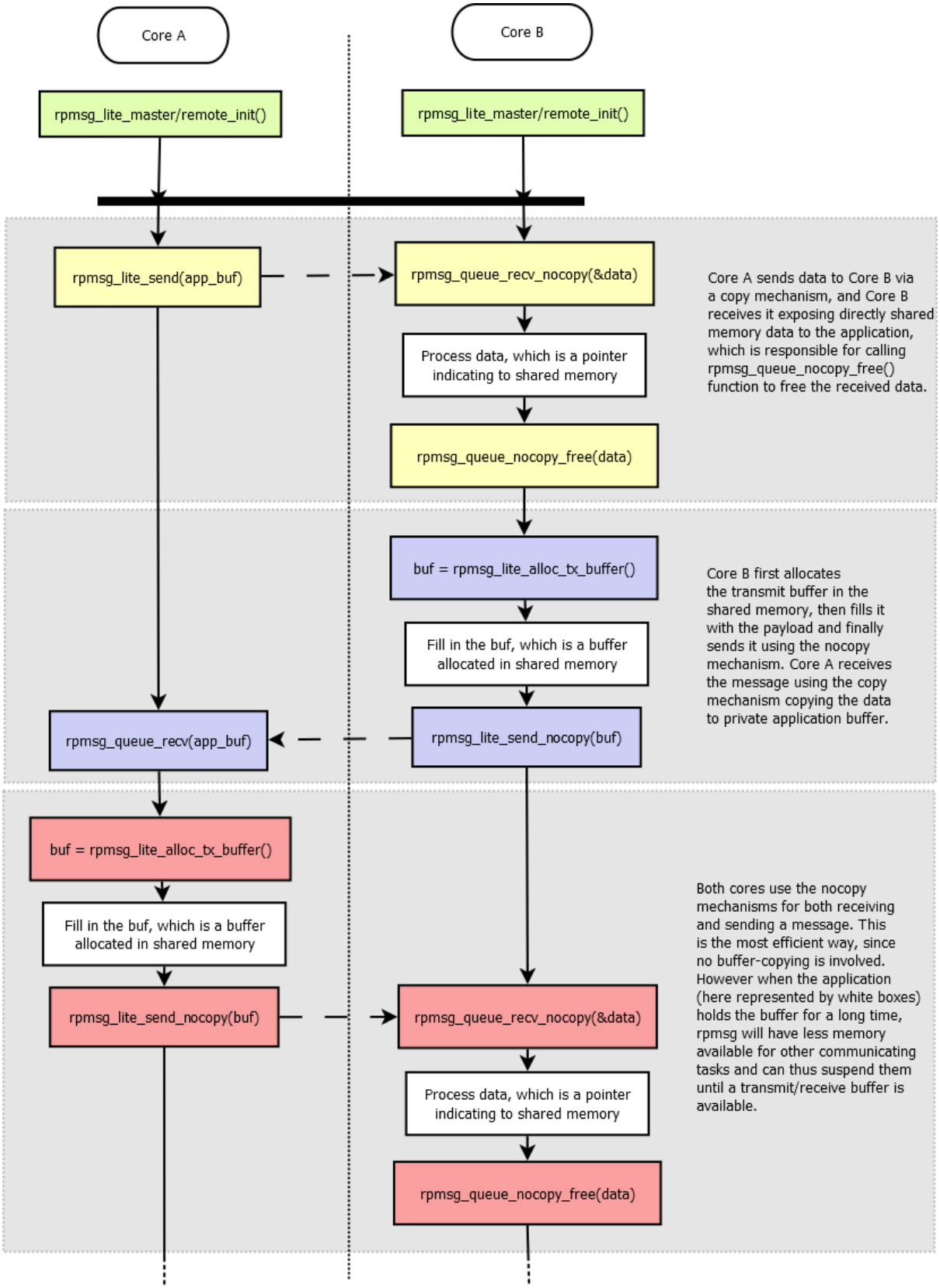
no-copy-send mechanism: This mechanism allows sending messages without the cost for copying data from the application buffer to the RPMsg/virtio buffer in the shared memory. The sequence of no-copy sending steps to be performed is as follows:

- 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_ALLOV	(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_ALLOV	(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).

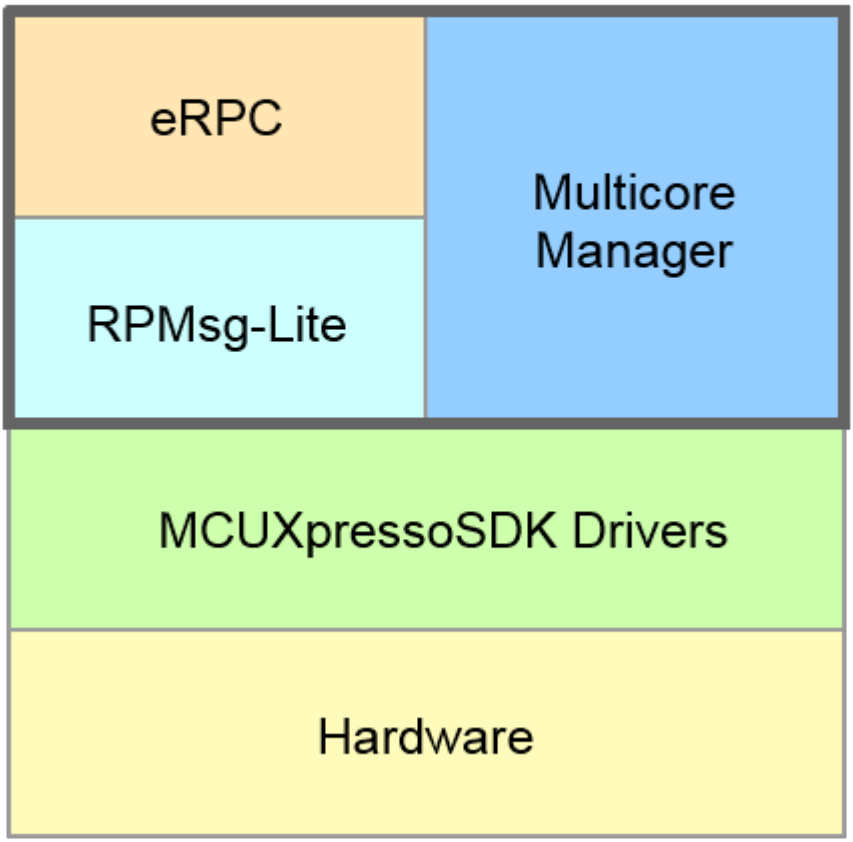
Multicore Manager (MCMGR)

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"

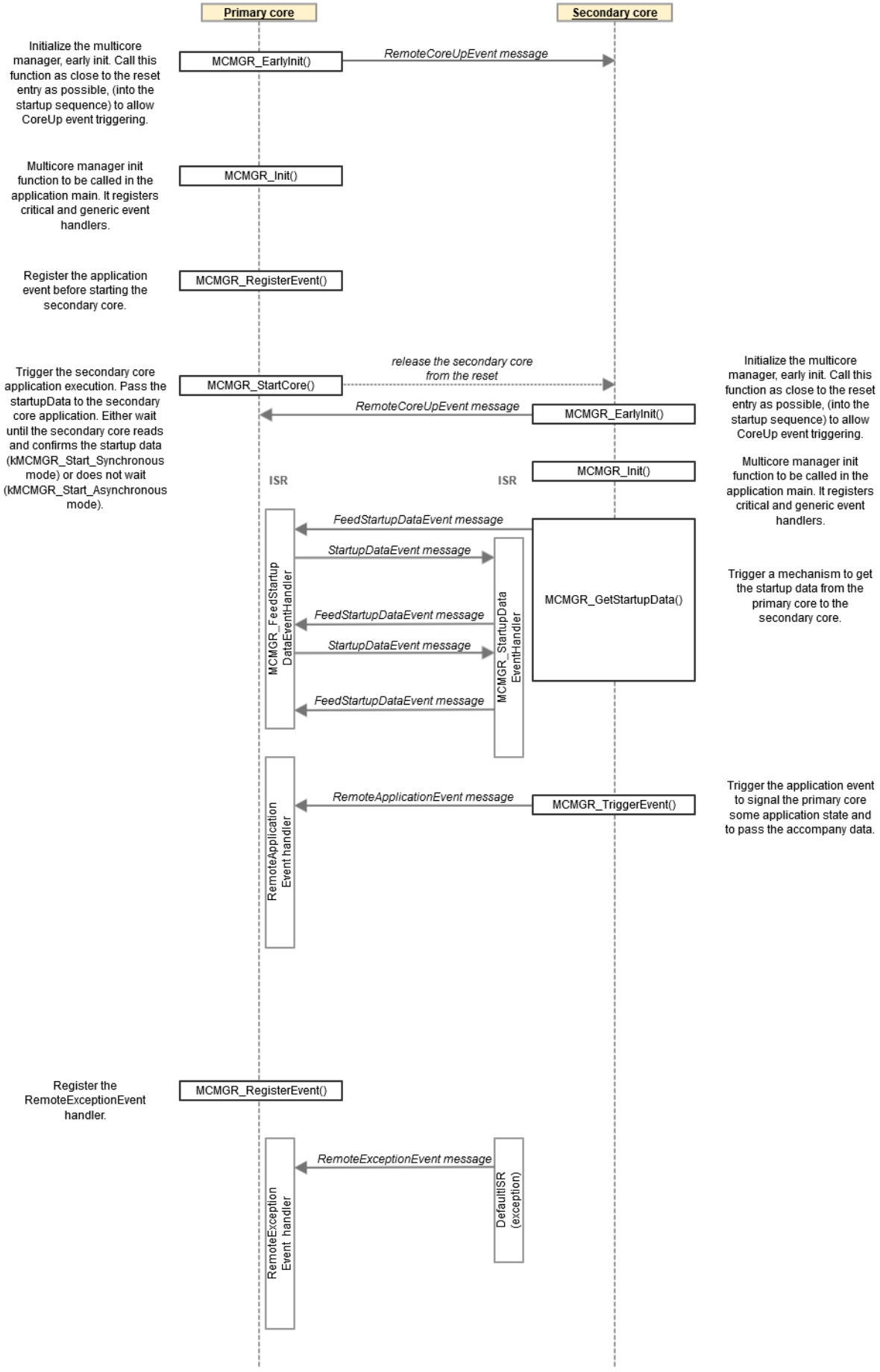
#define APP_RPMSG_READY_EVENT_DATA  (1)

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: PortingGuideTo_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:

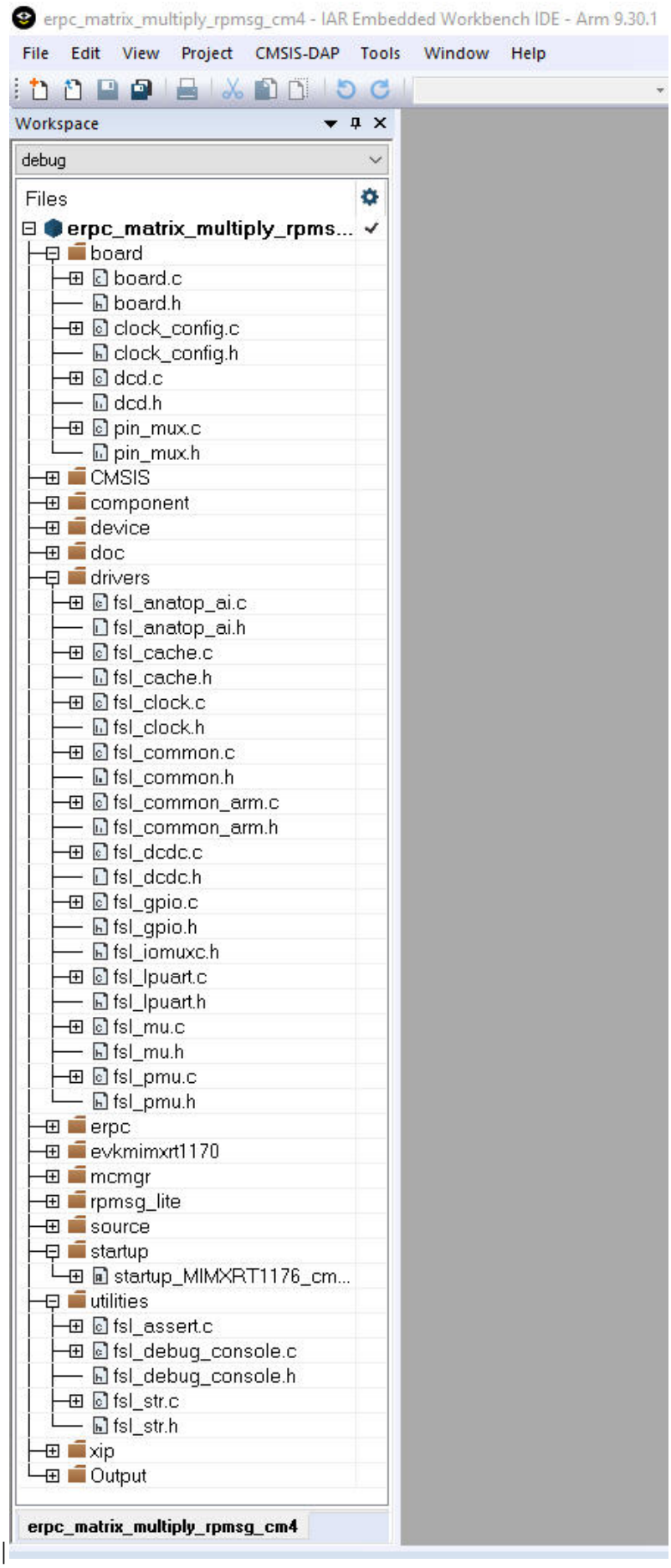
<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar

The project files for the eRPC server have the $_cm4$ 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

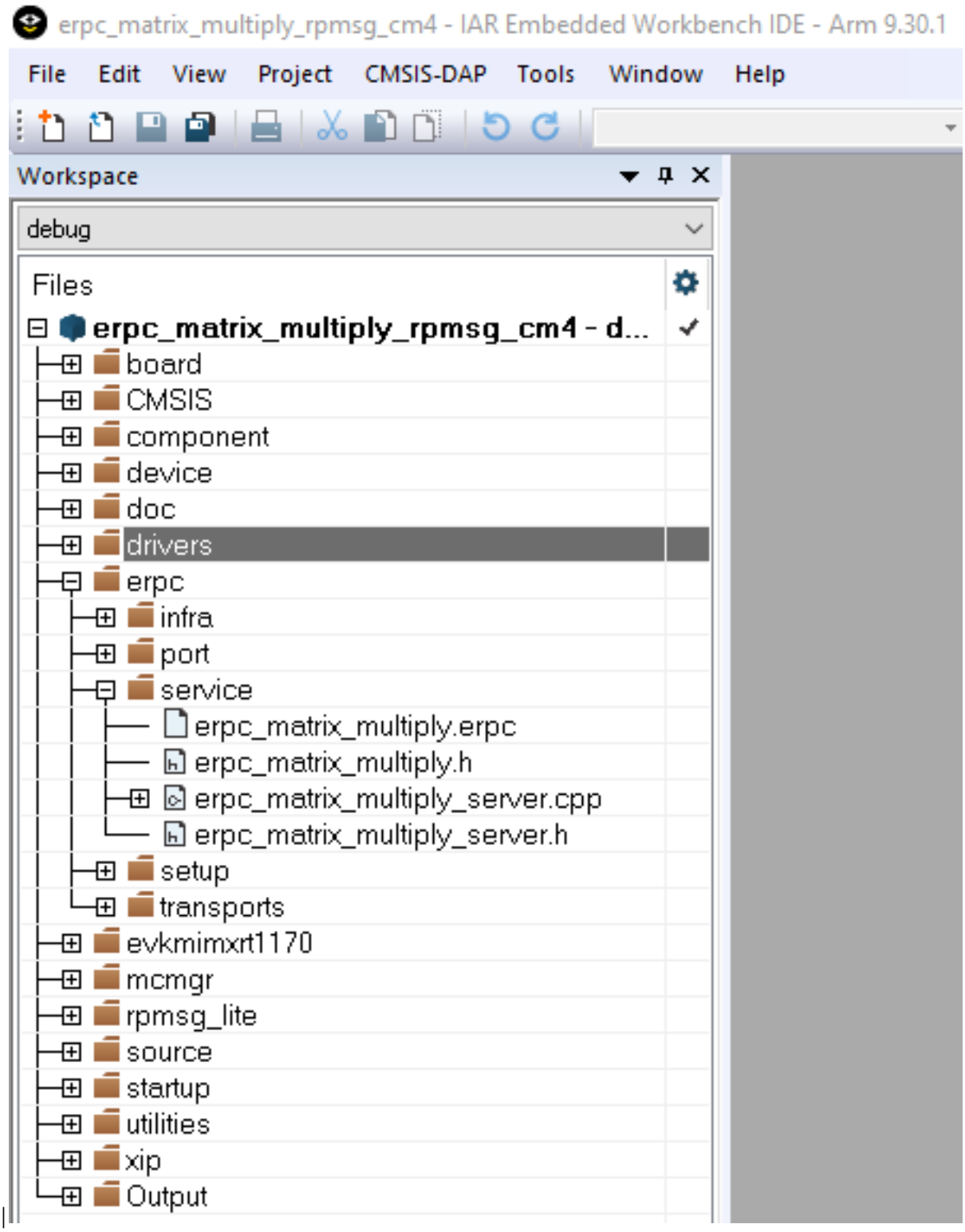
Server related generated files

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

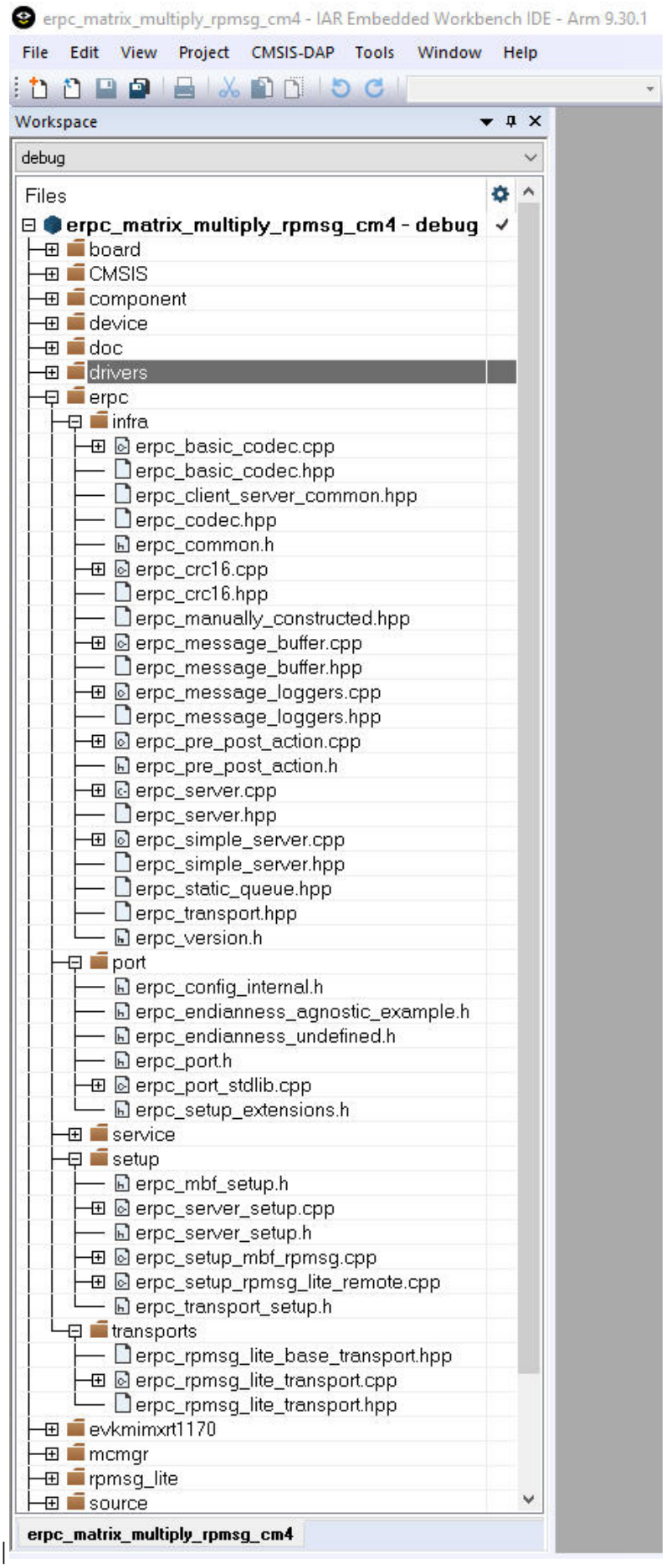
The eRPC infrastructure files are located in the following folder:

<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

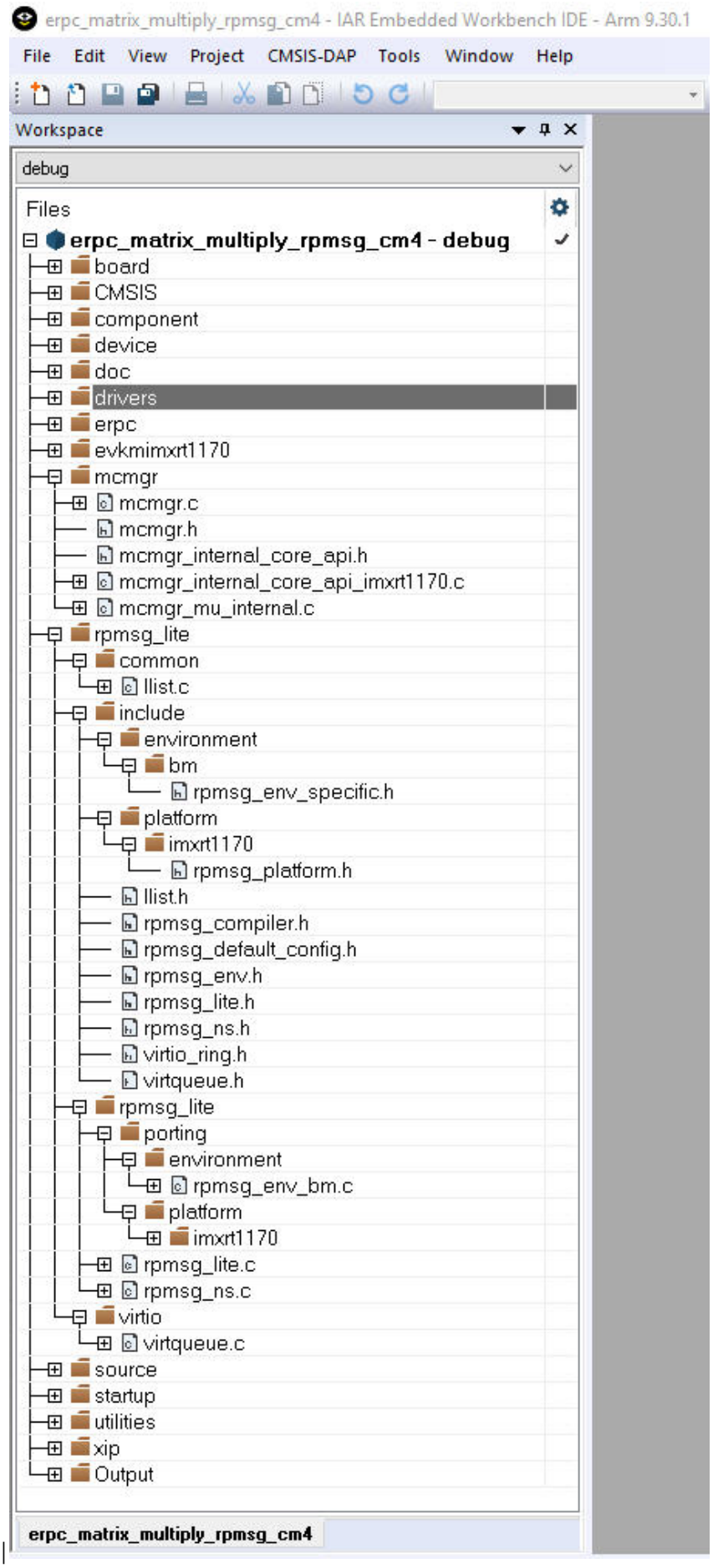
Server multicore infrastructure files

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

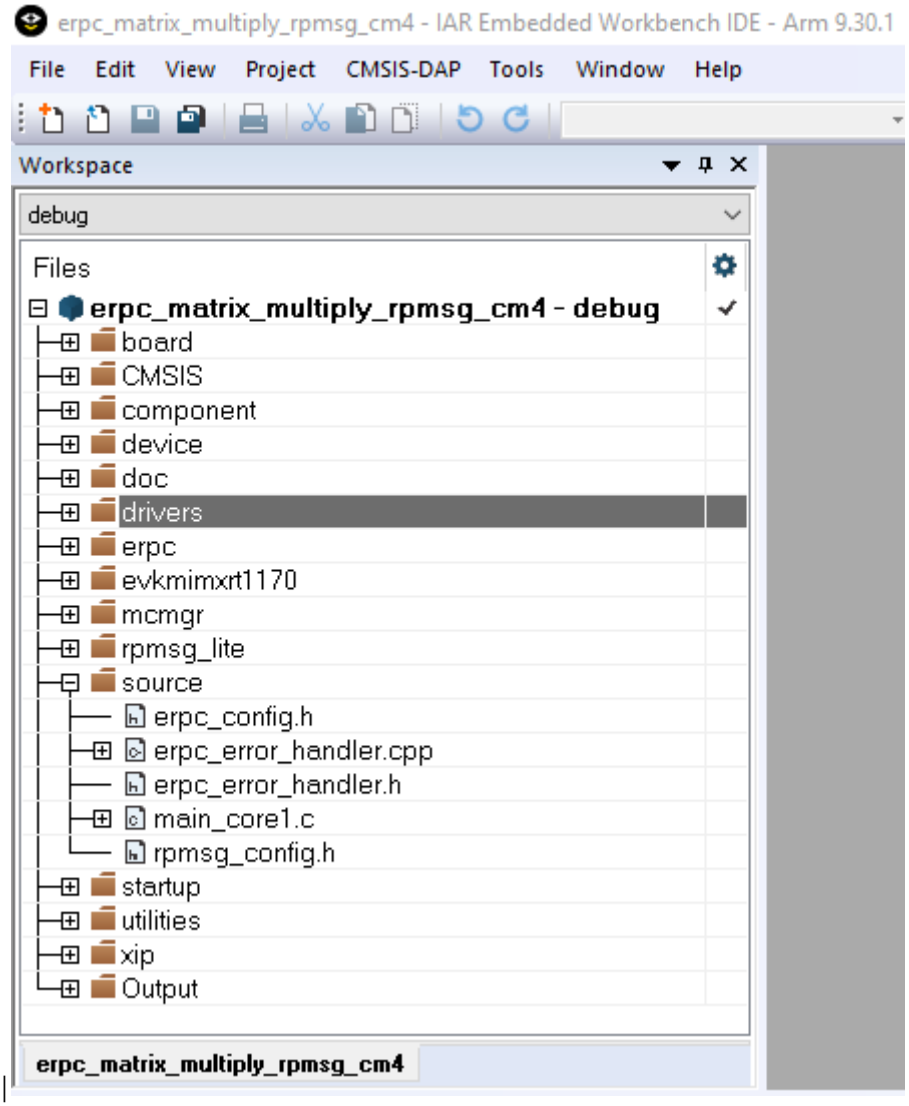
The main_core1.c file contains two functions:

- 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

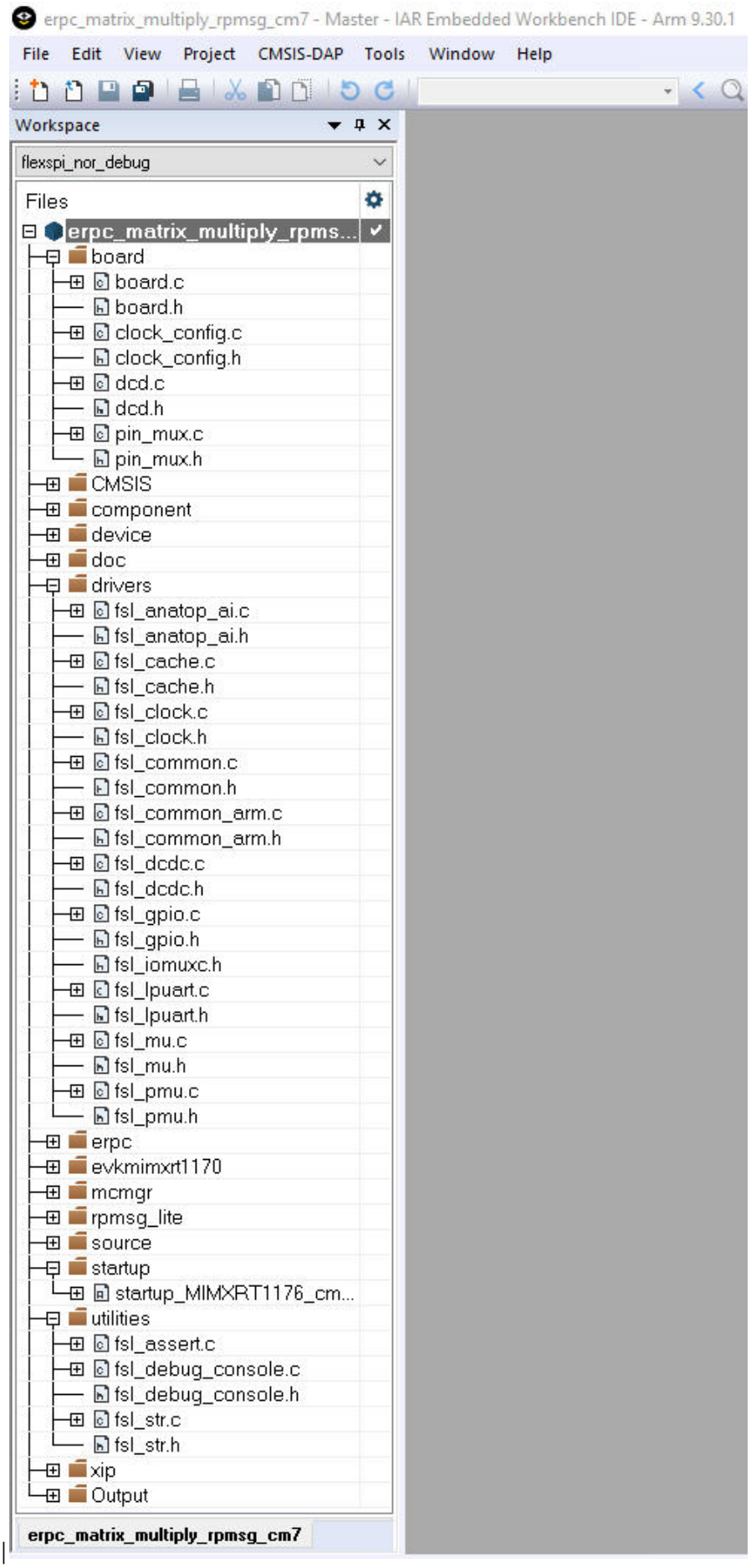
Project files for the eRPC client have the _cm7 suffix.

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

|

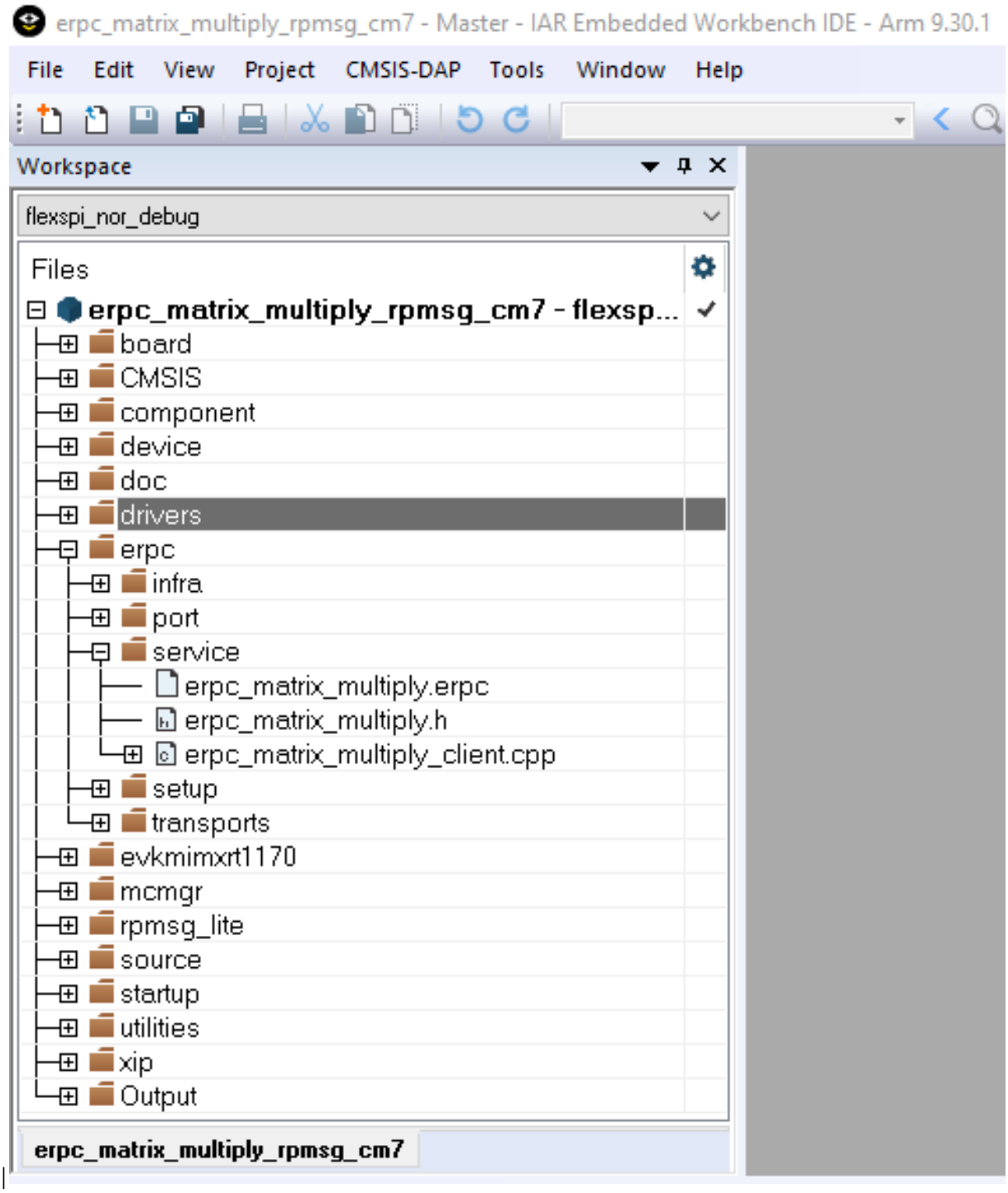
Parent topic: Multicore client application

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.

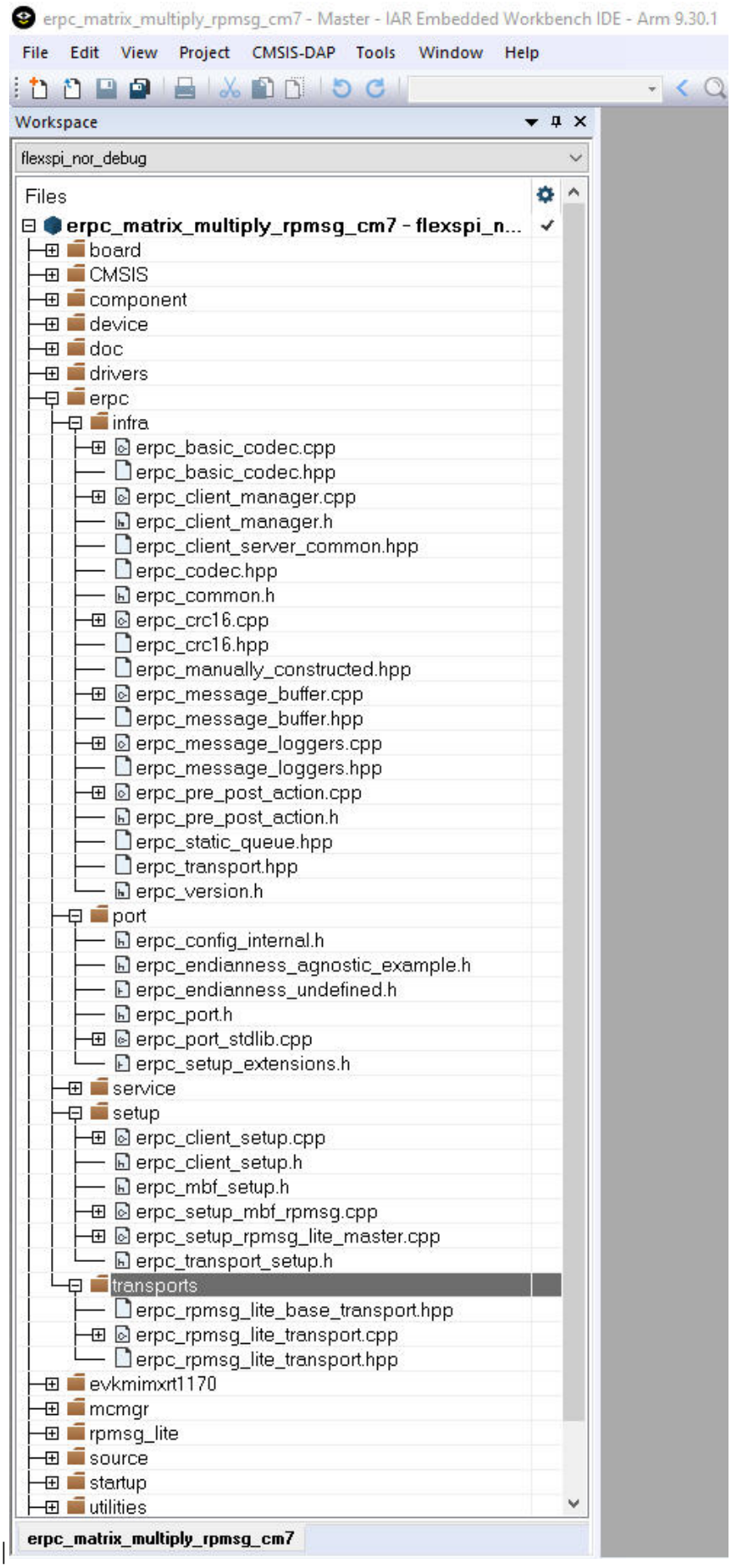
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

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

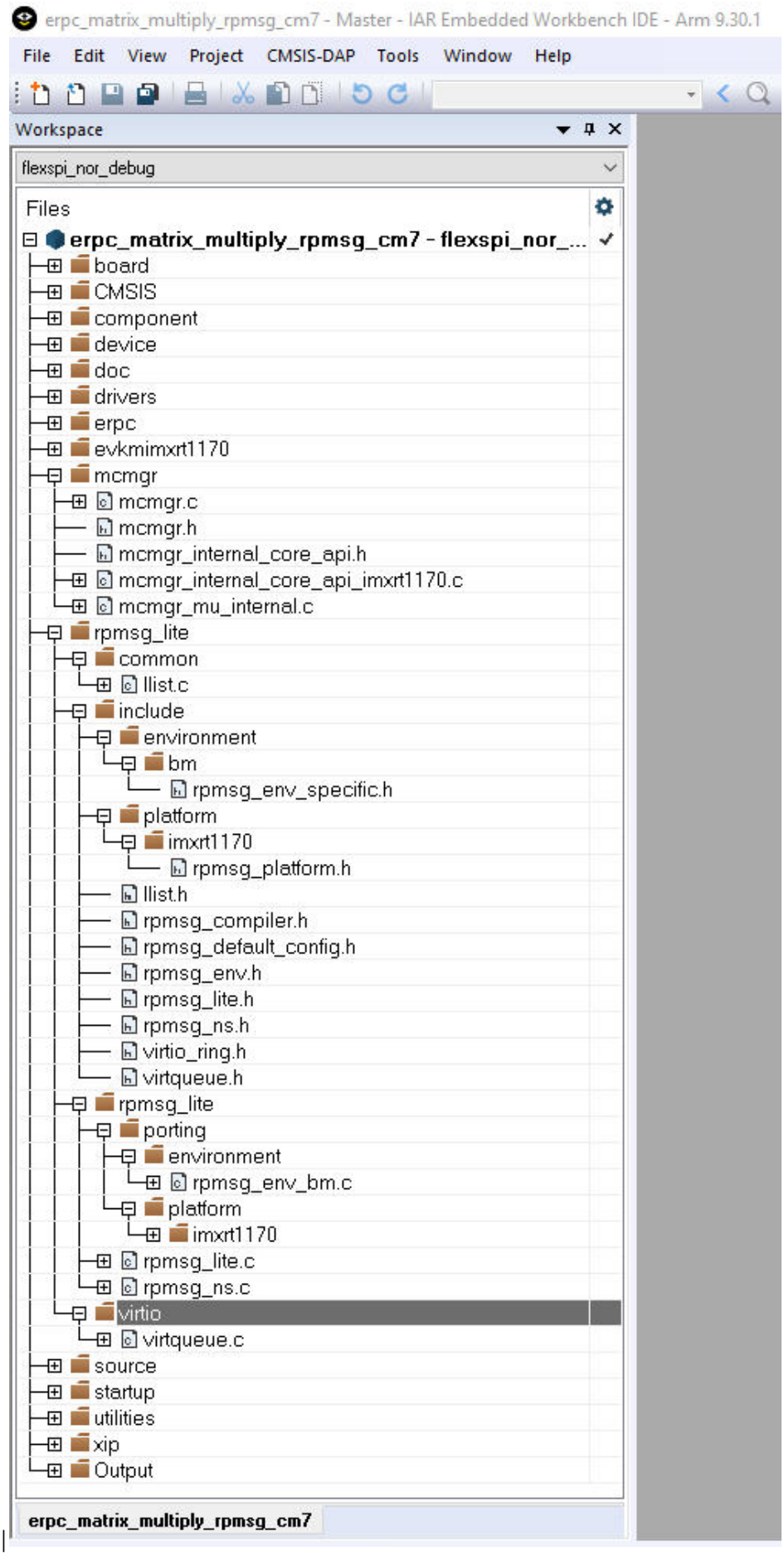
Client multicore infrastructure files

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.

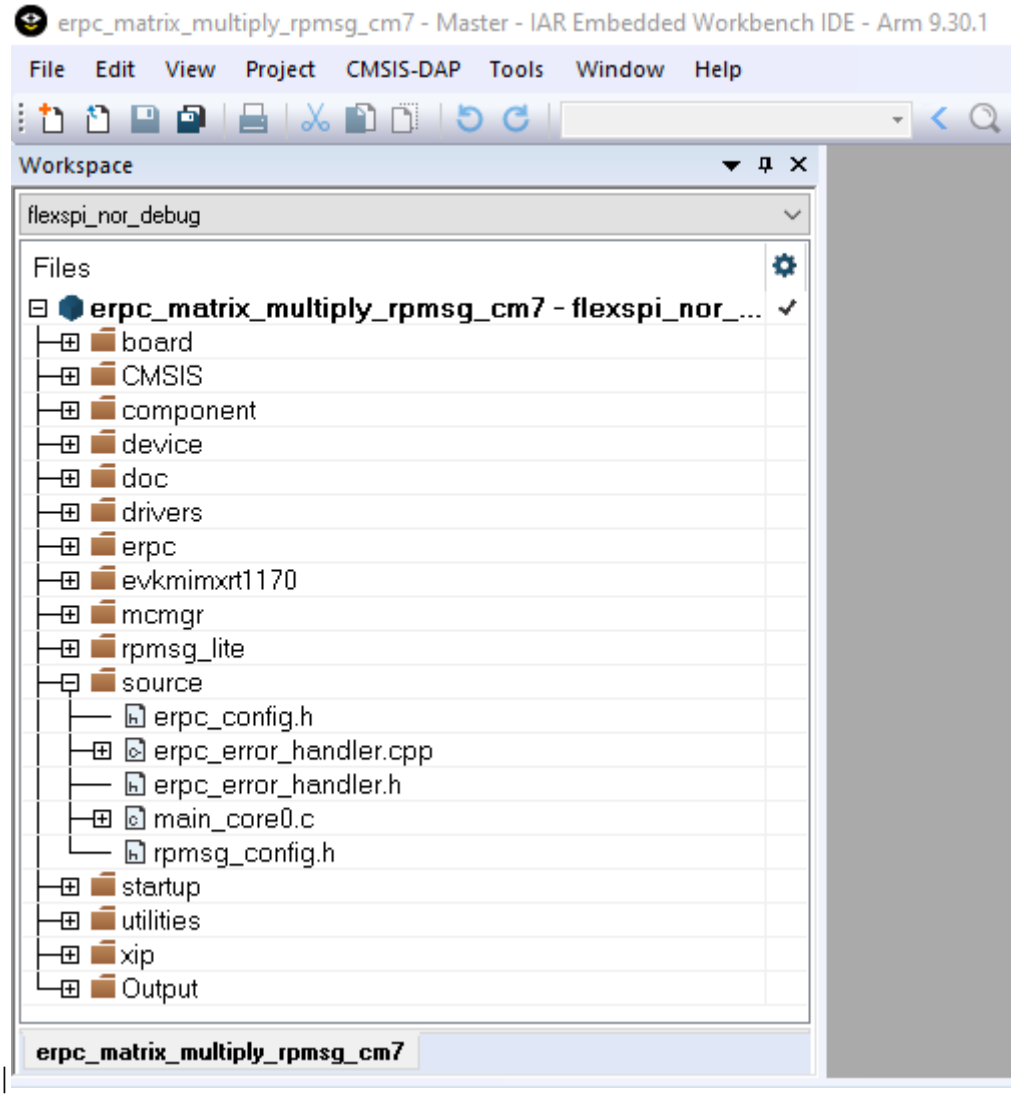
The matrix multiplication can be issued repeatedly, when pressing a software board button.

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;
  }
  ...
}
```

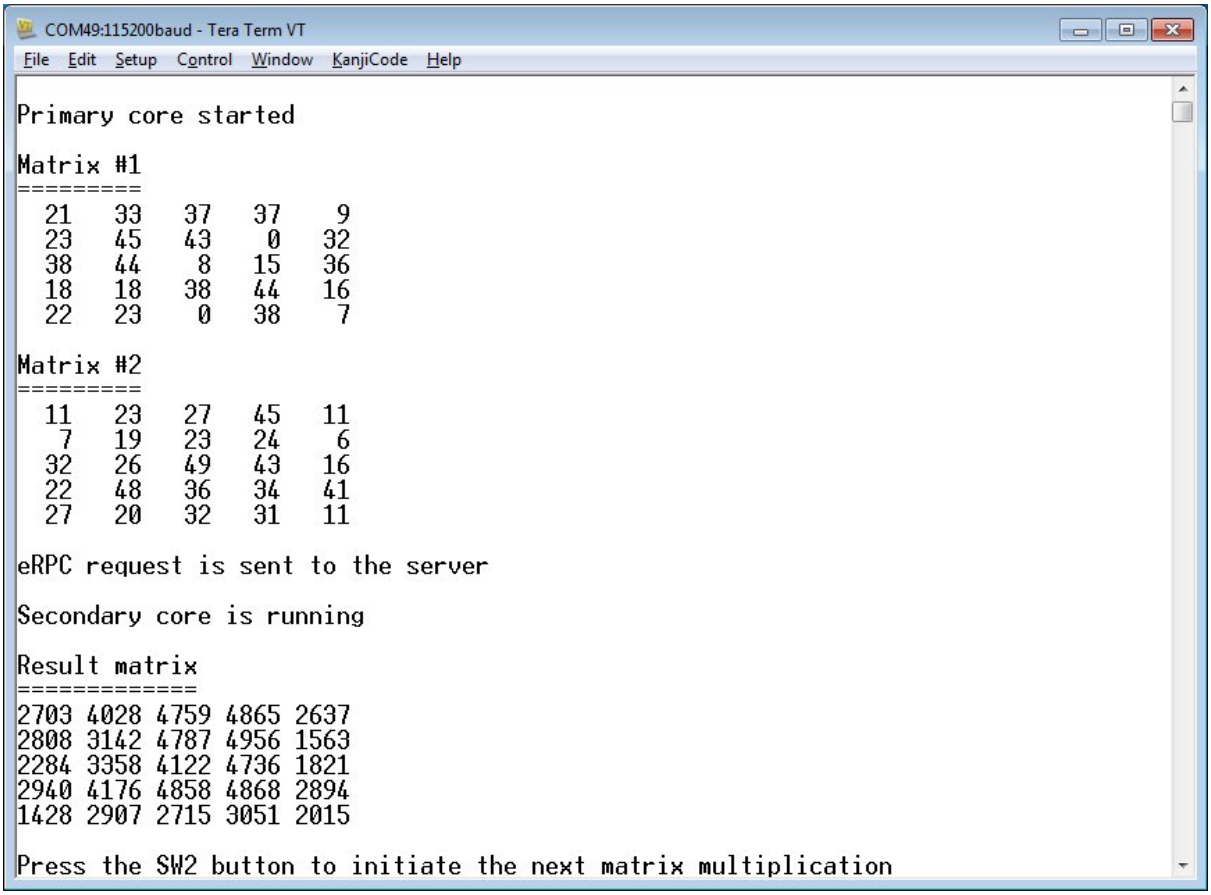
Parent topic: Multiprocessor client application

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:

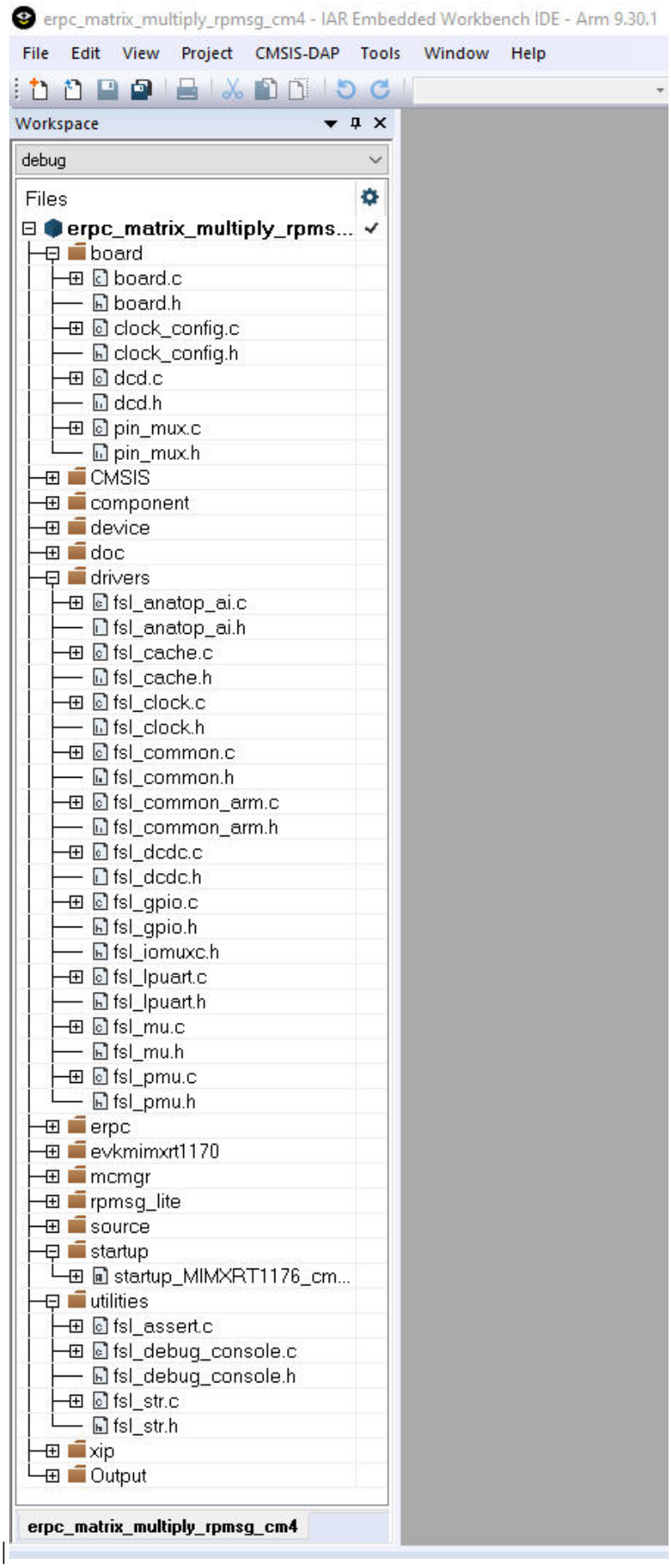
<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar

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

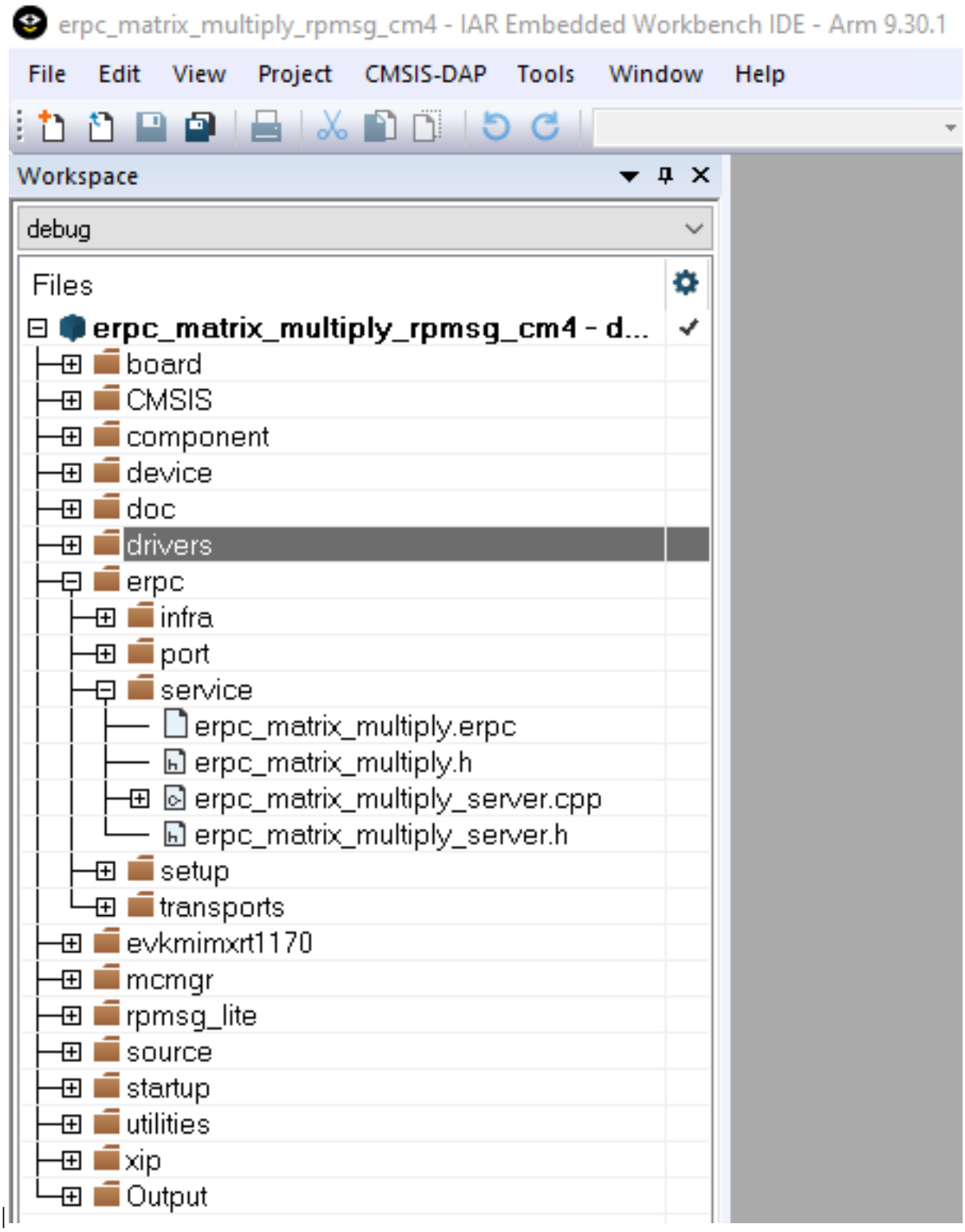
Server related generated files

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

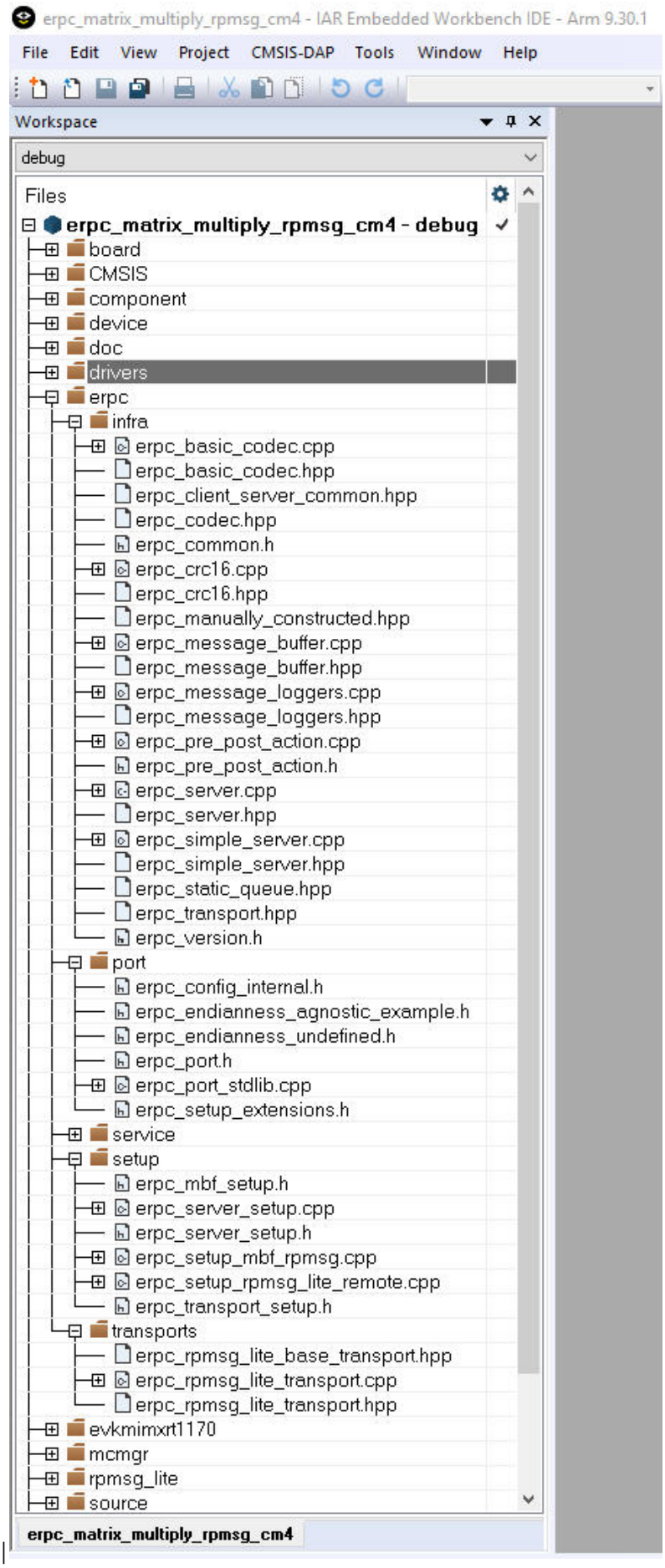
The eRPC infrastructure files are located in the following folder:

<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

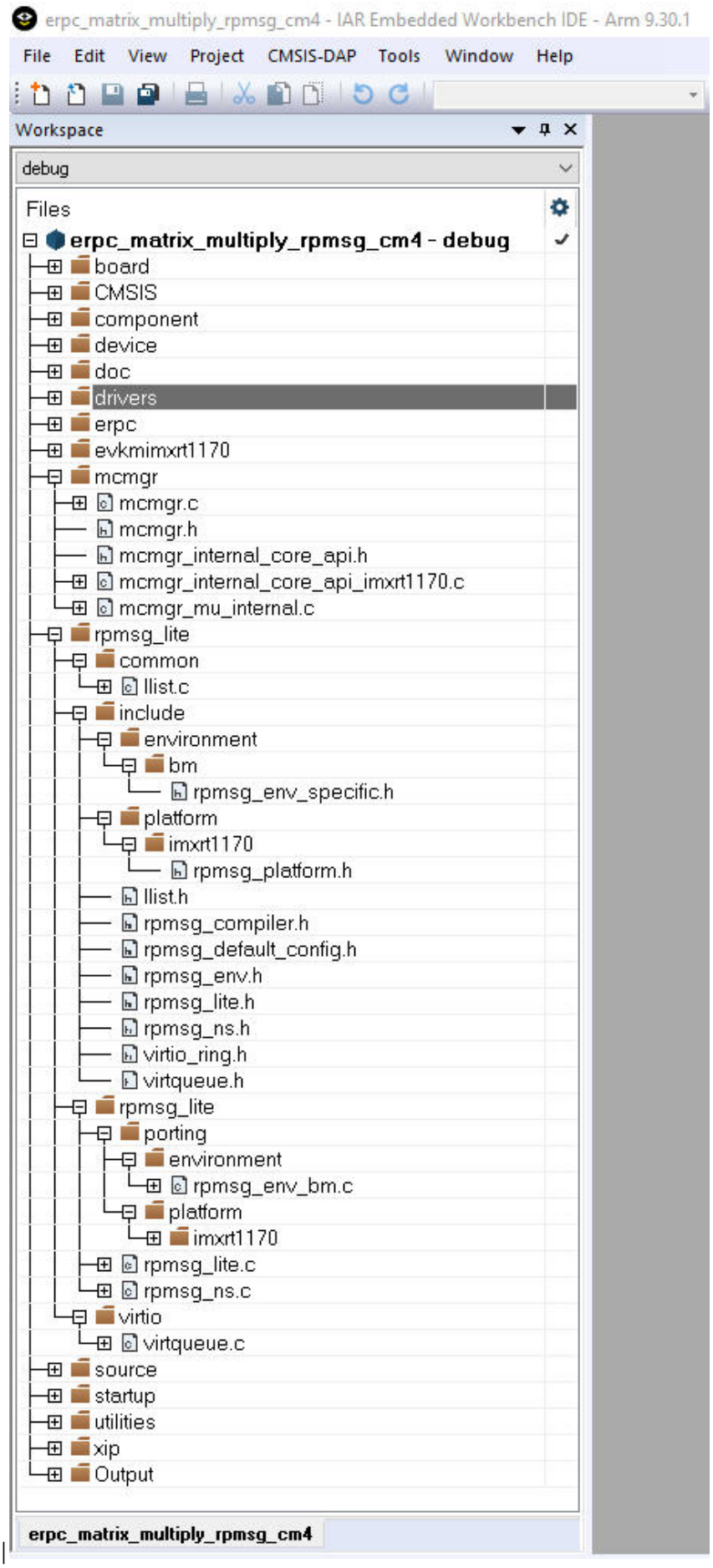
Server multicore infrastructure files

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

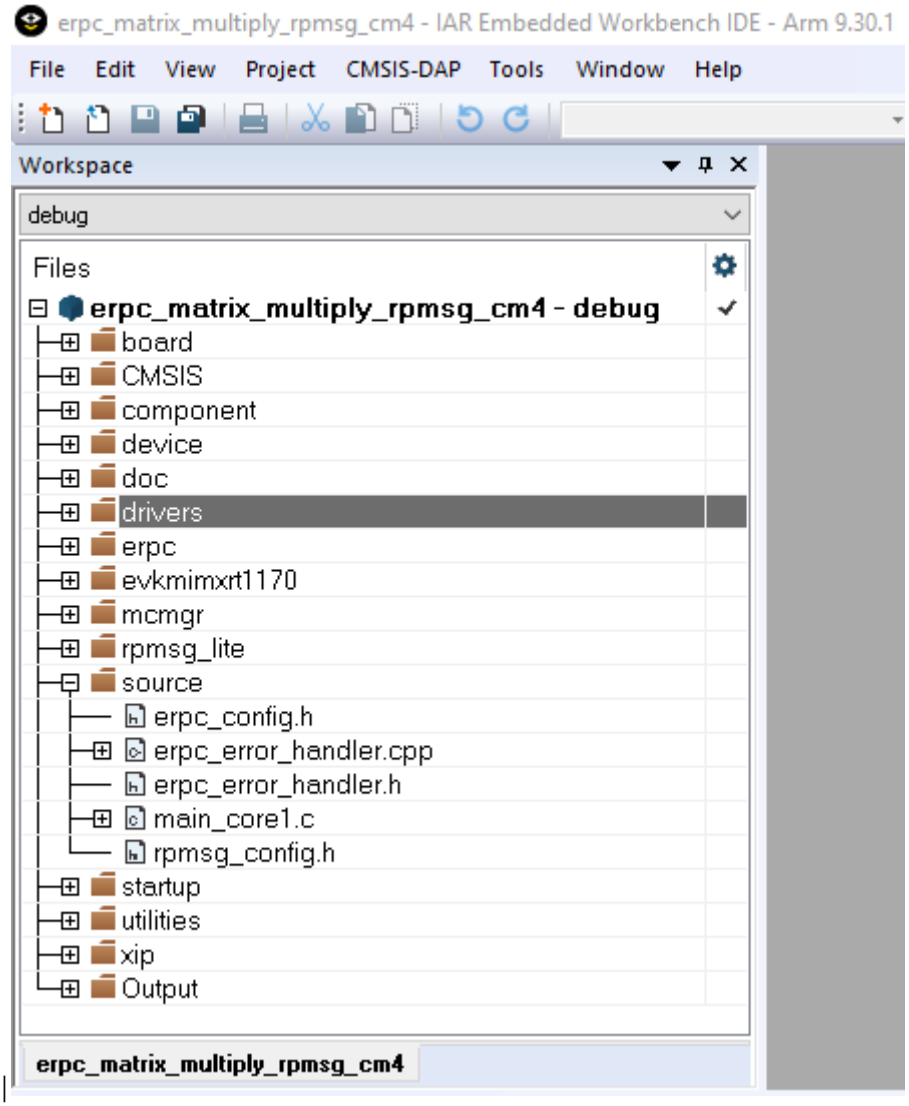
The main_core1.c file contains two functions:

- 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

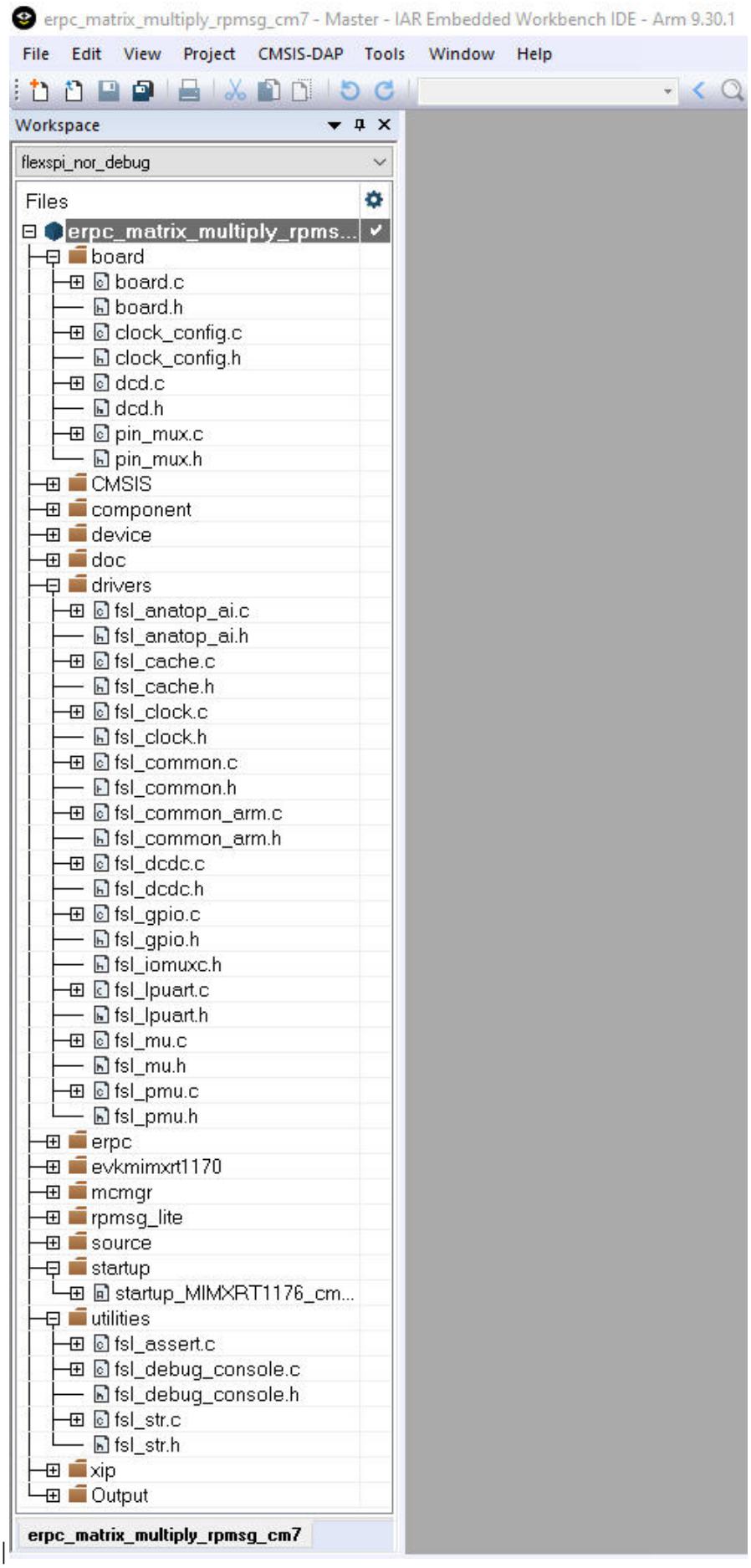
Project files for the eRPC client have the _cm7 suffix.

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

|

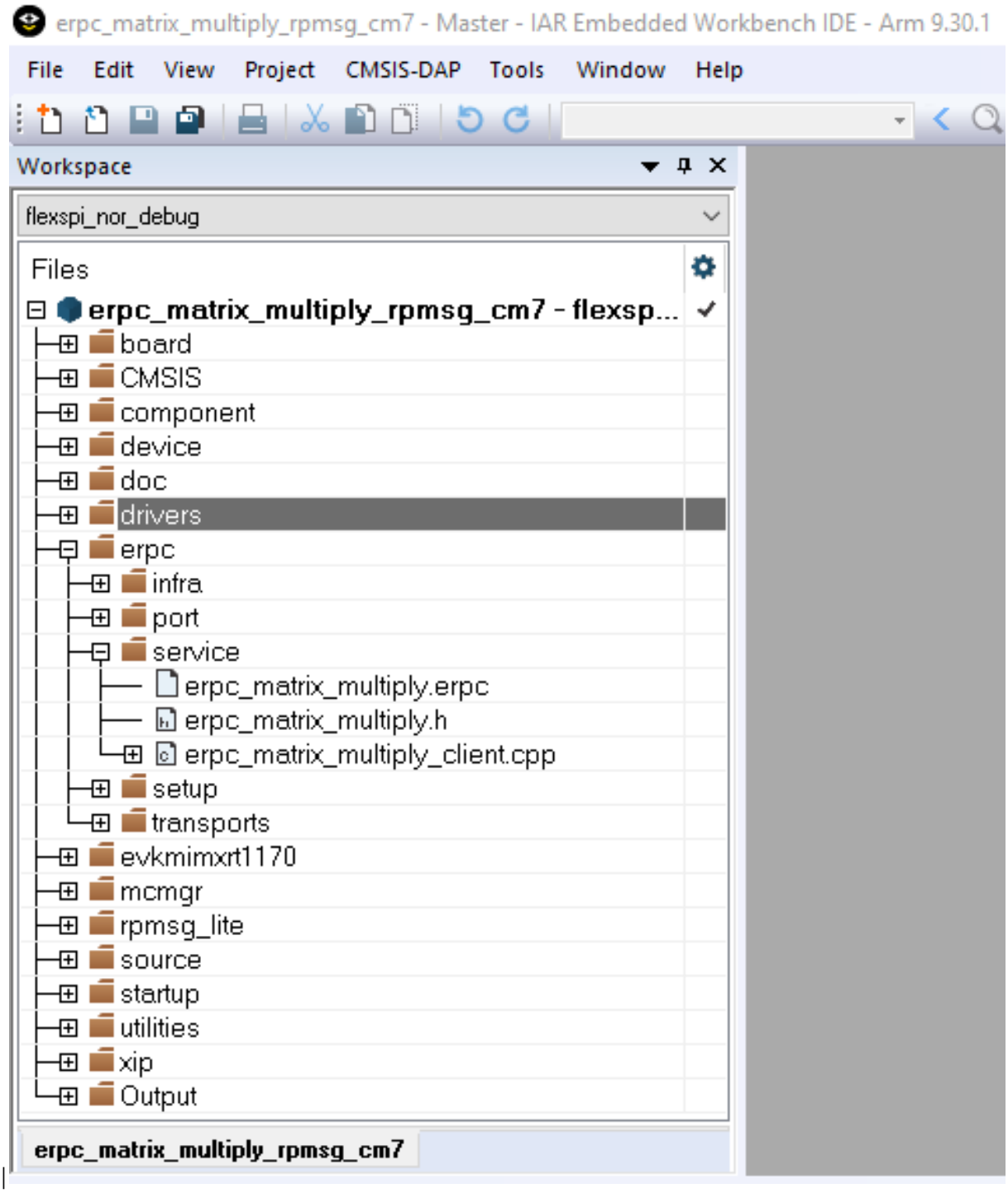
Parent topic: Multicore client application

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.

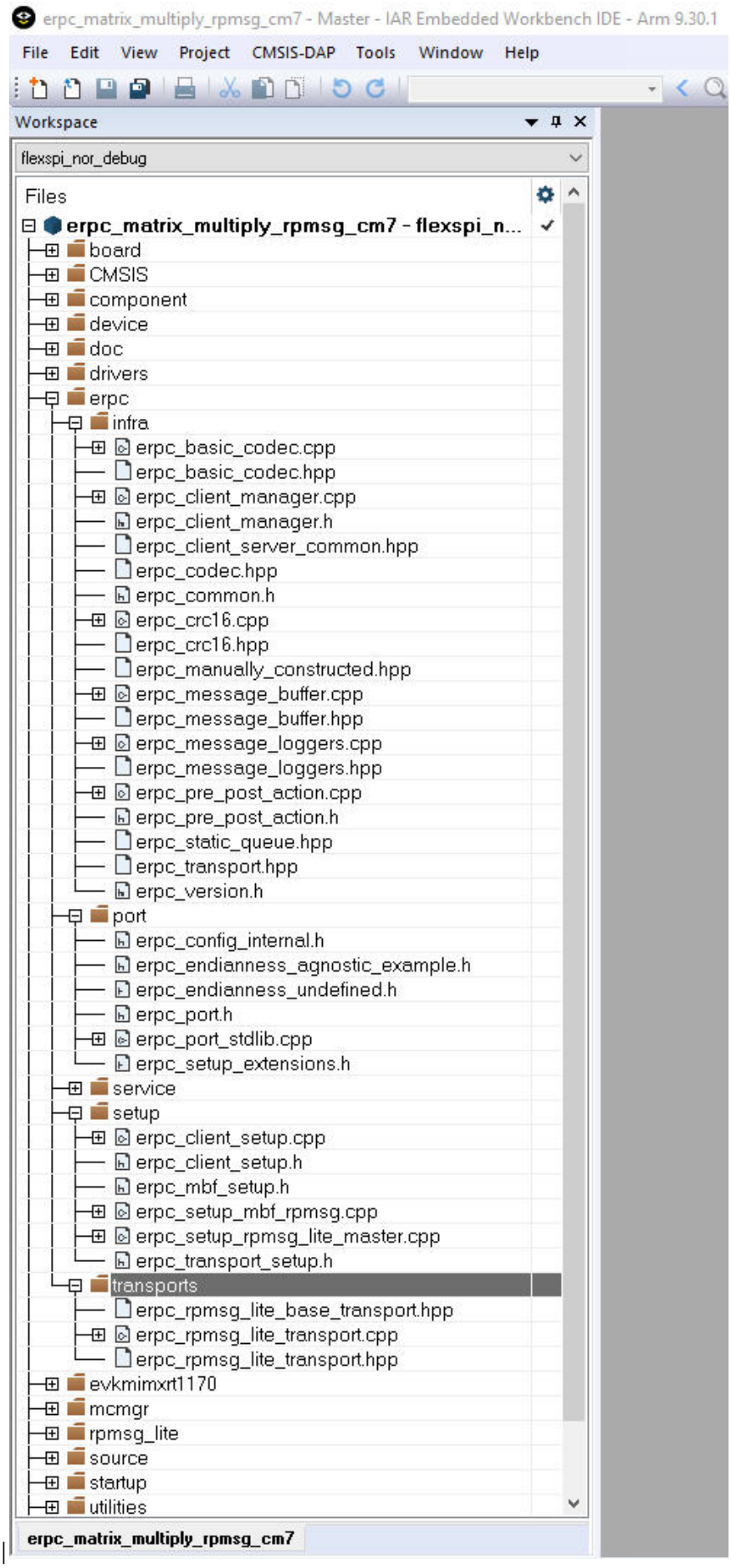
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

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

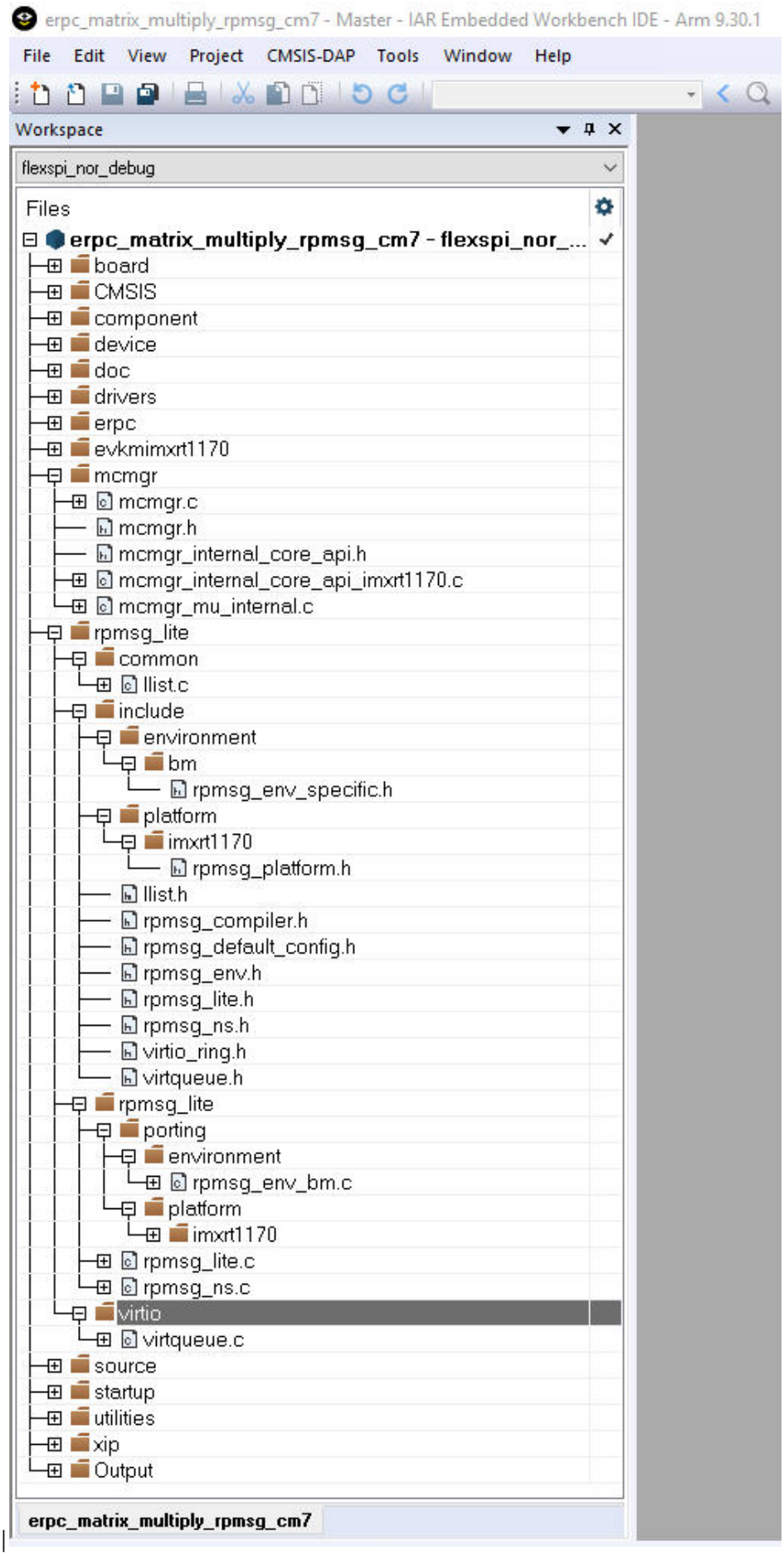
Client multicore infrastructure files

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.

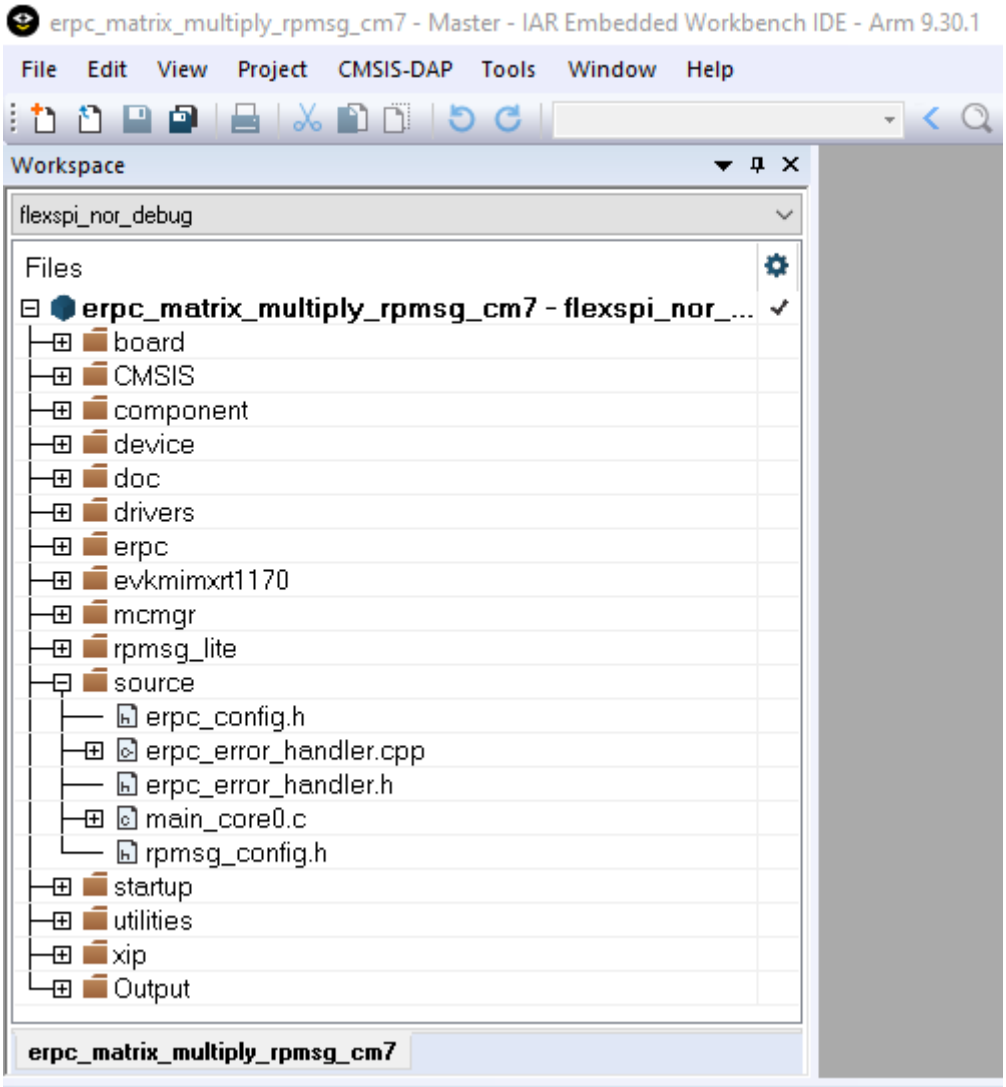
The matrix multiplication can be issued repeatedly, when pressing a software board button.

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;
  }
  ...
}
```

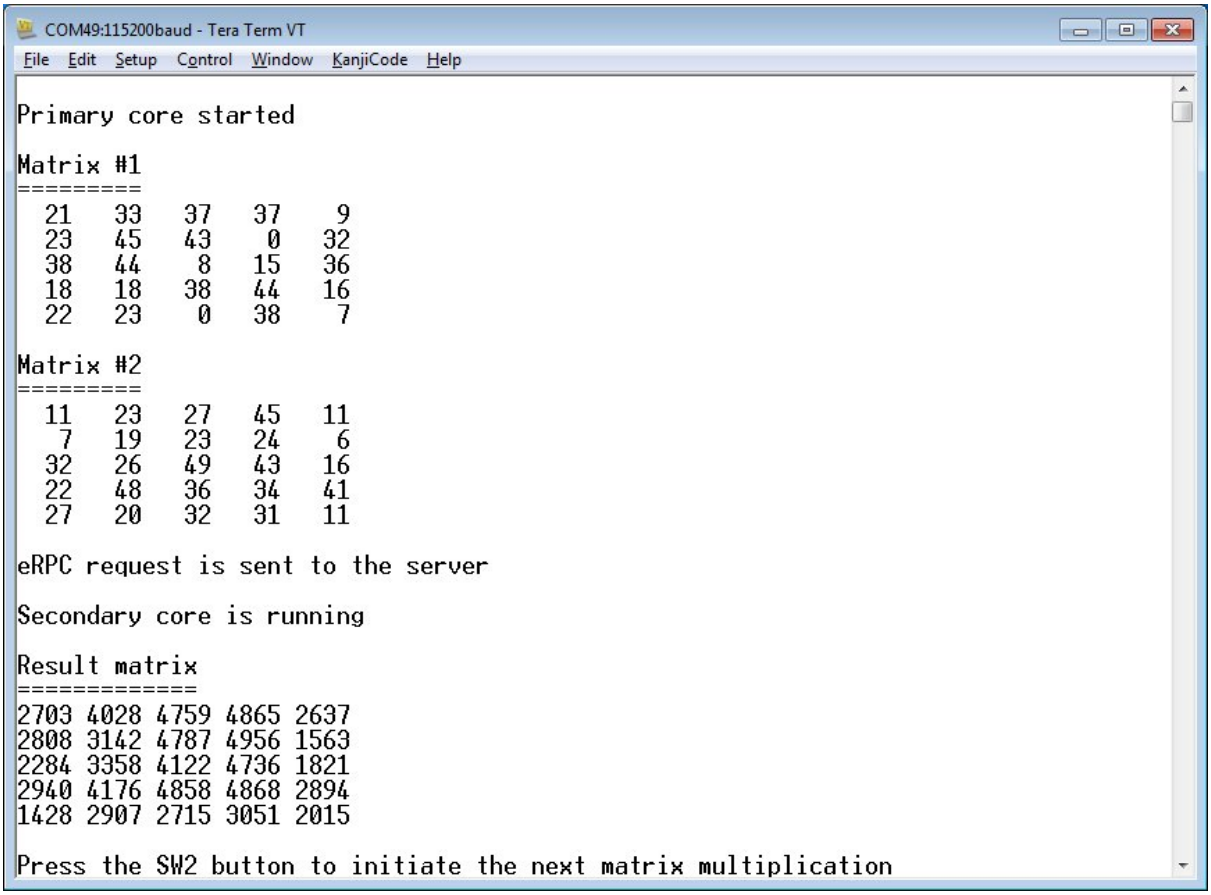
Parent topic: Multiprocessor client application

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

3.6 Multimedia

3.6.1 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
```

(continues on next page)

(continued from previous page)

```
[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

"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
******************************

[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'
[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

"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
```

(continues on next page)

(continued from previous page)

```
[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.

3.7 Wireless

3.7.1 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
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

Table 1 – continued from p

Features	Sub features
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
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 - using
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

Table 1 – continued from p

Features	Sub features
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)

Table 2 – continued from prev

Features	Sub features
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
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 3 – continued from prev

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.
Bluetootl	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

Com po- nent	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.
Blue tootl	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 4 – 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.

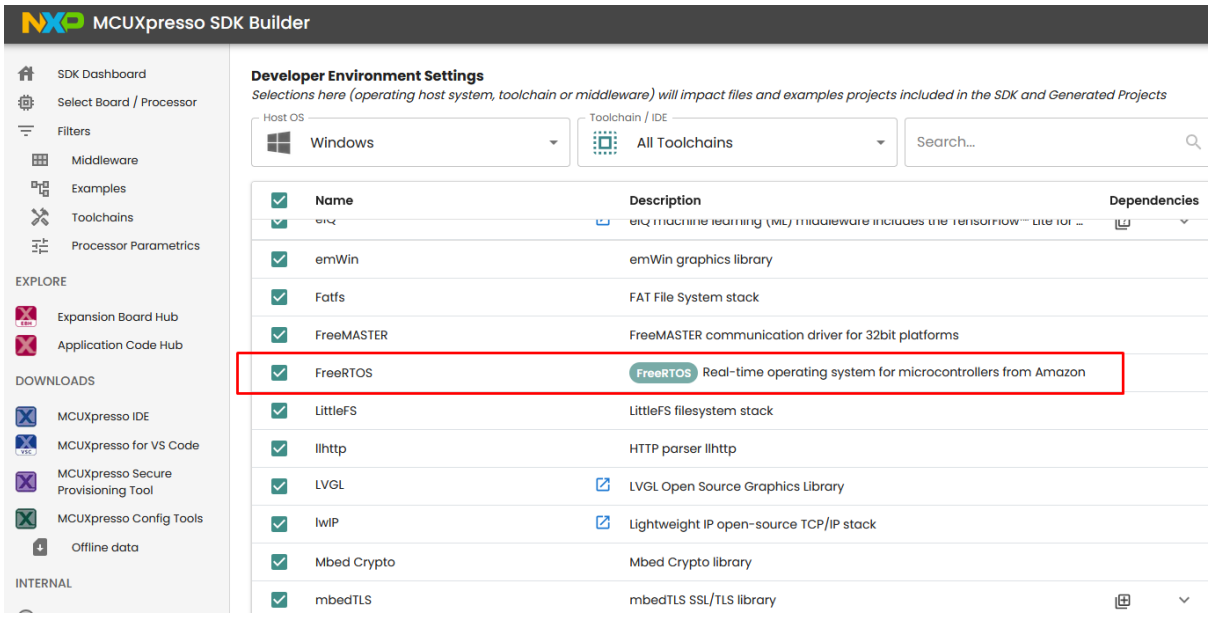
List of examples

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

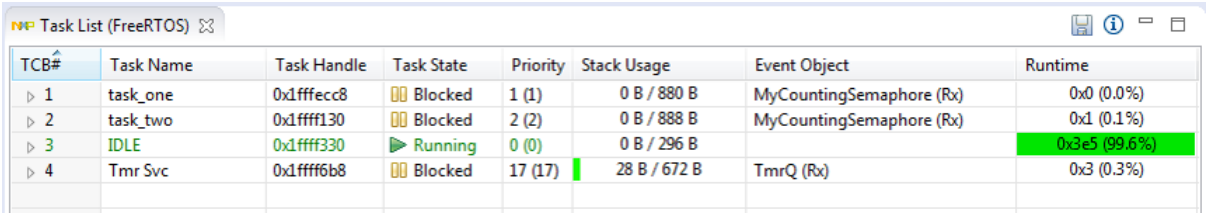
For information how to use the cmake and Kconfig based build and configuration system and how to build freertos_examples visit: MCUXpresso SDK documentation for Build And Configuration MCUXpresso SDK Getting Start Guide

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-evkmimxrt685.pdf — from *boards/RT/evkmimxrt685/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

Index

D

drivers_RT_RT600_MIMXRT685S.___pad0___ (*C var*), 578
drivers_RT_RT600_MIMXRT685S._acmp_channel_config (*C struct*), 213
drivers_RT_RT600_MIMXRT685S._acmp_channel_config.minusMuxInput (*C var*), 213
drivers_RT_RT600_MIMXRT685S._acmp_channel_config.negativePortInput (*C var*), 213
drivers_RT_RT600_MIMXRT685S._acmp_channel_config.plusMuxInput (*C var*), 213
drivers_RT_RT600_MIMXRT685S._acmp_channel_config.positivePortInput (*C var*), 213
drivers_RT_RT600_MIMXRT685S._acmp_config (*C struct*), 212
drivers_RT_RT600_MIMXRT685S._acmp_config.enableHighSpeed (*C var*), 212
drivers_RT_RT600_MIMXRT685S._acmp_config.enableInvertOutput (*C var*), 212
drivers_RT_RT600_MIMXRT685S._acmp_config.enablePinOut (*C var*), 213
drivers_RT_RT600_MIMXRT685S._acmp_config.hysteresisMode (*C var*), 212
drivers_RT_RT600_MIMXRT685S._acmp_config.offsetMode (*C var*), 212
drivers_RT_RT600_MIMXRT685S._acmp_config.useUnfilteredOutput (*C var*), 213
drivers_RT_RT600_MIMXRT685S._acmp_dac_config (*C struct*), 213
drivers_RT_RT600_MIMXRT685S._acmp_dac_config.DACValue (*C var*), 214
drivers_RT_RT600_MIMXRT685S._acmp_dac_config.enableOutput (*C var*), 214
drivers_RT_RT600_MIMXRT685S._acmp_dac_config.referenceVoltageSource (*C var*), 213
drivers_RT_RT600_MIMXRT685S._acmp_dac_work_mode (*C enum*), 211
drivers_RT_RT600_MIMXRT685S._acmp_dac_work_mode.kACMP_DACWorkHighSpeedMode (*C enumerator*), 211
drivers_RT_RT600_MIMXRT685S._acmp_dac_work_mode.kACMP_DACWorkLowSpeedMode (*C enumerator*), 211
drivers_RT_RT600_MIMXRT685S._acmp_discrete_mode_config (*C struct*), 214
drivers_RT_RT600_MIMXRT685S._acmp_discrete_mode_config.enableNegativeChannelDiscreteMode (*C var*), 214
drivers_RT_RT600_MIMXRT685S._acmp_discrete_mode_config.enablePositiveChannelDiscreteMode (*C var*), 214
drivers_RT_RT600_MIMXRT685S._acmp_filter_config (*C struct*), 213
drivers_RT_RT600_MIMXRT685S._acmp_filter_config.filterCount (*C var*), 213
drivers_RT_RT600_MIMXRT685S._acmp_filter_config.filterPeriod (*C var*), 213
drivers_RT_RT600_MIMXRT685S._acmp_hysteresis_mode (*C enum*), 210
drivers_RT_RT600_MIMXRT685S._acmp_hysteresis_mode.kACMP_HysteresisLevel0 (*C enumerator*), 210
drivers_RT_RT600_MIMXRT685S._acmp_hysteresis_mode.kACMP_HysteresisLevel1 (*C enumerator*), 211
drivers_RT_RT600_MIMXRT685S._acmp_hysteresis_mode.kACMP_HysteresisLevel2 (*C enumerator*), 211
drivers_RT_RT600_MIMXRT685S._acmp_hysteresis_mode.kACMP_HysteresisLevel3 (*C enumerator*), 211
drivers_RT_RT600_MIMXRT685S._acmp_interrupt_enable (*C enum*), 210
drivers_RT_RT600_MIMXRT685S._acmp_interrupt_enable.kACMP_OutputFallingInterruptEnable (*C enumerator*), 210
drivers_RT_RT600_MIMXRT685S._acmp_interrupt_enable.kACMP_OutputRisingInterruptEnable (*C enumerator*), 210
drivers_RT_RT600_MIMXRT685S._acmp_offset_mode (*C enum*), 210

drivers_RT_RT600_MIMXRT685S._acmp_offset_mode.kACMP_OffsetLevel0 (*C enumerator*), 210
drivers_RT_RT600_MIMXRT685S._acmp_offset_mode.kACMP_OffsetLevel1 (*C enumerator*), 210
drivers_RT_RT600_MIMXRT685S._acmp_port_input (*C enum*), 211
drivers_RT_RT600_MIMXRT685S._acmp_port_input.kACMP_PortInputFromDAC (*C enumerator*), 211
drivers_RT_RT600_MIMXRT685S._acmp_port_input.kACMP_PortInputFromMux (*C enumerator*), 211
drivers_RT_RT600_MIMXRT685S._acmp_reference_voltage_source (*C enum*), 211
drivers_RT_RT600_MIMXRT685S._acmp_reference_voltage_source.kACMP_VrefSourceVin1 (*C enumerator*), 211
drivers_RT_RT600_MIMXRT685S._acmp_reference_voltage_source.kACMP_VrefSourceVin2 (*C enumerator*), 211
drivers_RT_RT600_MIMXRT685S._acmp_status_flags (*C enum*), 210
drivers_RT_RT600_MIMXRT685S._acmp_status_flags.kACMP_OutputAssertEventFlag (*C enumerator*), 210
drivers_RT_RT600_MIMXRT685S._acmp_status_flags.kACMP_OutputFallingEventFlag (*C enumerator*), 210
drivers_RT_RT600_MIMXRT685S._acmp_status_flags.kACMP_OutputRisingEventFlag (*C enumerator*), 210
drivers_RT_RT600_MIMXRT685S._audio_pll_mult (*C enum*), 249
drivers_RT_RT600_MIMXRT685S._audio_pll_mult.kCLOCK_AudioPllMult16 (*C enumerator*), 249
drivers_RT_RT600_MIMXRT685S._audio_pll_mult.kCLOCK_AudioPllMult17 (*C enumerator*), 249
drivers_RT_RT600_MIMXRT685S._audio_pll_mult.kCLOCK_AudioPllMult18 (*C enumerator*), 249
drivers_RT_RT600_MIMXRT685S._audio_pll_mult.kCLOCK_AudioPllMult19 (*C enumerator*), 250
drivers_RT_RT600_MIMXRT685S._audio_pll_mult.kCLOCK_AudioPllMult20 (*C enumerator*), 250
drivers_RT_RT600_MIMXRT685S._audio_pll_mult.kCLOCK_AudioPllMult21 (*C enumerator*), 250
drivers_RT_RT600_MIMXRT685S._audio_pll_mult.kCLOCK_AudioPllMult22 (*C enumerator*), 250
drivers_RT_RT600_MIMXRT685S._audio_pll_src (*C enum*), 249
drivers_RT_RT600_MIMXRT685S._audio_pll_src.kCLOCK_AudioPllFFroDiv2 (*C enumerator*), 249
drivers_RT_RT600_MIMXRT685S._audio_pll_src.kCLOCK_AudioPllNone (*C enumerator*), 249
drivers_RT_RT600_MIMXRT685S._audio_pll_src.kCLOCK_AudioPllSFroClk (*C enumerator*), 249
drivers_RT_RT600_MIMXRT685S._audio_pll_src.kCLOCK_AudioPllXtalIn (*C enumerator*), 249
drivers_RT_RT600_MIMXRT685S._body_bias_mode (*C enum*), 569
drivers_RT_RT600_MIMXRT685S._body_bias_mode.kPmu_Fbb (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S._body_bias_mode.kPmu_Nbb (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._body_bias_mode.kPmu_Rbb (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S._cache64_config (*C struct*), 218
drivers_RT_RT600_MIMXRT685S._cache64_config.boundaryAddr (*C var*), 218
drivers_RT_RT600_MIMXRT685S._cache64_policy (*C enum*), 217
drivers_RT_RT600_MIMXRT685S._cache64_policy.kCACHE64_PolicyNonCacheable (*C enumerator*), 217
drivers_RT_RT600_MIMXRT685S._cache64_policy.kCACHE64_PolicyWriteBack (*C enumerator*), 217
drivers_RT_RT600_MIMXRT685S._cache64_policy.kCACHE64_PolicyWriteThrough (*C enumerator*), 217
drivers_RT_RT600_MIMXRT685S._casper_algo_t (*C enum*), 220
drivers_RT_RT600_MIMXRT685S._casper_algo_t.kCASPER_ECC_P256 (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_algo_t.kCASPER_ECC_P384 (*C enumerator*), 221
drivers_RT_RT600_MIMXRT685S._casper_algo_t.kCASPER_ECC_P521 (*C enumerator*), 221
drivers_RT_RT600_MIMXRT685S._casper_operation (*C enum*), 219

drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpAdd64 (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpCompare (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpCompareFast (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpCopy (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpDouble64 (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpFill (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpMul6464FullSum (*C enumerator*), 219
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpMul6464NoSum (*C enumerator*), 219
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpMul6464Reduce (*C enumerator*), 219
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpMul6464Sum (*C enumerator*), 219
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpRemask (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpRSub64 (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpShiftLeft32 (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpShiftRight32 (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpSub64 (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpXor64 (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._casper_operation.kCASPER_OpZero (*C enumerator*), 220
drivers_RT_RT600_MIMXRT685S._clock_attach_id (*C enum*), 233
drivers_RT_RT600_MIMXRT685S._clock_attach_id.k32K_WAKE_CLK_to_DMIC_CLK (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_CLKOUT (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_CTIMER0 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_CTIMER1 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_CTIMER2 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_CTIMER3 (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_CTIMER4 (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_DMIC_CLK (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_FLEXCOMM0 (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_FLEXCOMM1 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_FLEXCOMM2 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_FLEXCOMM3 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_FLEXCOMM4 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_FLEXCOMM5 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_FLEXCOMM6 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_FLEXCOMM7 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_FLEXCOMM14 (*C enu-*

merator), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_FLEXCOMM15 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_MCLK_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUDIO_PLL_to_SCT_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX0_PLL_to_ACMP_CLK (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX0_PLL_to_ADC_CLK (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX0_PLL_to_CLKOUT (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX0_PLL_to_FLEXSPI_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX0_PLL_to_SCT_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX0_PLL_to_SDIO0_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX0_PLL_to_SDIO1_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX1_PLL_to_ACMP_CLK (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX1_PLL_to_ADC_CLK (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX1_PLL_to_CLKOUT (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX1_PLL_to_FLEXSPI_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX1_PLL_to_SCT_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX1_PLL_to_SDIO0_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kAUX1_PLL_to_SDIO1_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kDSP_MAIN_to_CLKOUT (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kDSP_PLL_to_CLKOUT (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kDSP_PLL_to_DSP_MAIN_CLK (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_DIV2_to_AUDIO_PLL (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_DIV2_to_SYS_PLL (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_DIV4_to_MAIN_CLK (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_ACMP_CLK (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_ADC_CLK (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_CLKOUT (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_CTIMER0 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_CTIMER1 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_CTIMER2 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_CTIMER3 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_CTIMER4 (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_DMIC_CLK (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_DSP_MAIN_CLK (*C enumera-*

tor), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_ESPI_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_FLEXCOMM0 (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_FLEXCOMM1 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_FLEXCOMM2 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_FLEXCOMM3 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_FLEXCOMM4 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_FLEXCOMM5 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_FLEXCOMM6 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_FLEXCOMM7 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_FLEXCOMM14 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_FLEXCOMM15 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_FLEXSPI_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_I3C_CLK (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_MAIN_CLK (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_MCLK_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_SCT_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_SDIO0_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFFRO_to_SDIO1_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFRG_to_FLEXCOMM0 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFRG_to_FLEXCOMM1 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFRG_to_FLEXCOMM2 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFRG_to_FLEXCOMM3 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFRG_to_FLEXCOMM4 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFRG_to_FLEXCOMM5 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFRG_to_FLEXCOMM6 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFRG_to_FLEXCOMM7 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFRG_to_FLEXCOMM14 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kFRG_to_FLEXCOMM15 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kHCLK_to_OSTIMER_CLK (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kI3C_CLK_to_I3C_TC_CLK (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_DIV32_to_32KHZWAKE_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_ADC_CLK (*C enumerator*), 235

drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_CLKOUT (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_CTIMER0 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_CTIMER1 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_CTIMER2 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_CTIMER3 (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_CTIMER4 (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_DMIC_CLK (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_DSP_MAIN_CLK (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_I3C_TC_CLK (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_MAIN_CLK (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_OSTIMER_CLK (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_SYSTICK_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_UTICK_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_WDT0_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kLPOSC_to_WDT1_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_DIV_to_SYSTICK_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_ACMP_CLK (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_CLKOUT (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_CTIMER0 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_CTIMER1 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_CTIMER2 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_CTIMER3 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_CTIMER4 (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_FLEXSPI_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_I3C_CLK (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_SCT_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_SDIO0_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_SDIO1_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_CLK_to_USB_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_PLL_to_ADC_CLK (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_PLL_to_CLKOUT (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_PLL_to_DSP_MAIN_CLK (*C enumerator*), 234

drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_PLL_to_FLEXSPI_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_PLL_to_MAIN_CLK (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_PLL_to_SCT_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_PLL_to_SDIO0_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMAIN_PLL_to_SDIO1_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_CTIMER0 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_CTIMER1 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_CTIMER2 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_CTIMER3 (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_CTIMER4 (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_DMIC_CLK (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_FLEXCOMM0 (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_FLEXCOMM1 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_FLEXCOMM2 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_FLEXCOMM3 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_FLEXCOMM4 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_FLEXCOMM5 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_FLEXCOMM6 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_FLEXCOMM7 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_FLEXCOMM14 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kMASTER_CLK_to_FLEXCOMM15 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_32KHZWAKE_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_ACMP_CLK (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_AUDIO_PLL (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_CLKOUT (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_CTIMER0 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_CTIMER1 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_CTIMER2 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_CTIMER3 (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_CTIMER4 (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_DMIC_CLK (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_ESPI_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_FLEXCOMM0 (*C enumerator*), 241

drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_FLEXCOMM1 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_FLEXCOMM2 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_FLEXCOMM3 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_FLEXCOMM4 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_FLEXCOMM5 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_FLEXCOMM6 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_FLEXCOMM7 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_FLEXCOMM14 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_FLEXCOMM15 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_FLEXSPI_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_I3C_CLK (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_I3C_TC_CLK (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_MCLK_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_OSTIMER_CLK (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_SCT_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_SDIO0_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_SDIO1_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_SYS_PLL (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_SYSTICK_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_USB_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_UTICK_CLK (*C enumerator*), 237
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_WDT0_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kNONE_to_WDT1_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kOSC32K_to_32KHZWAKE_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kOSC32K_to_CLKOUT (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kOSC32K_to_DSP_MAIN_CLK (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kOSC32K_to_MAIN_CLK (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kOSC32K_to_OSTIMER_CLK (*C enumerator*), 236
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kOSC32K_to_SYSTICK_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_ACMP_CLK (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_ADC_CLK (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_AUDIO_PLL (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_CLKOUT (*C enumerator*), 235

drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_CTIMER0 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_CTIMER1 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_CTIMER2 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_CTIMER3 (*C enumerator*), 245
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_CTIMER4 (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_DMIC_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_DSP_MAIN_CLK (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_FLEXCOMM0 (*C enumerator*), 240
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_FLEXCOMM1 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_FLEXCOMM2 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_FLEXCOMM3 (*C enumerator*), 241
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_FLEXCOMM4 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_FLEXCOMM5 (*C enumerator*), 242
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_FLEXCOMM6 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_FLEXCOMM7 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_FLEXCOMM14 (*C enumerator*), 243
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_FLEXCOMM15 (*C enumerator*), 244
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_MAIN_CLK (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_SYS_PLL (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kSFRO_to_SYSTICK_CLK (*C enumerator*), 238
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kXTALIN_CLK_to_ADC_CLK (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kXTALIN_CLK_to_AUDIO_PLL (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kXTALIN_CLK_to_CLKOUT (*C enumerator*), 235
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kXTALIN_CLK_to_DSP_MAIN_CLK (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kXTALIN_CLK_to_MAIN_CLK (*C enumerator*), 234
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kXTALIN_CLK_to_SYS_PLL (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_attach_id.kXTALIN_CLK_to_USB_CLK (*C enumerator*), 239
drivers_RT_RT600_MIMXRT685S._clock_audio_pll_config (*C struct*), 263
drivers_RT_RT600_MIMXRT685S._clock_div_name (*C enum*), 246
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivAcmpClk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivAdcClk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivAudioPllClk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivAux0PllClk (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivAux1PllClk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivClockOut (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivDmicClk (*C enumerator*), 247

drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivDspCpuClk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivDspPllClk (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivDspRamClk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivFlexspiClk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivI3cClk (*C enumerator*), 248
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivI3cSlowClk (*C enumerator*), 248
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivI3cTcClk (*C enumerator*), 248
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivMainPllClk (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivMclkClk (*C enumerator*), 248
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivPfc0Clk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivPfc1Clk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivPllFrgClk (*C enumerator*), 248
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivSctClk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivSdio0Clk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivSdio1Clk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivSysCpuAhbClk (*C enumerator*), 246
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivSystickClk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_div_name.kCLOCK_DivUsbHsFclk (*C enumerator*), 247
drivers_RT_RT600_MIMXRT685S._clock_ffro_freq (*C enum*), 248
drivers_RT_RT600_MIMXRT685S._clock_ffro_freq.kCLOCK_Ffro48M (*C enumerator*), 248
drivers_RT_RT600_MIMXRT685S._clock_ffro_freq.kCLOCK_Ffro60M (*C enumerator*), 248
drivers_RT_RT600_MIMXRT685S._clock_frg_clk_config (*C struct*), 263
drivers_RT_RT600_MIMXRT685S._clock_ip_name (*C enum*), 225
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Acmp0 (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Adc0 (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Casper (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Crc (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Ct32b0 (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Ct32b1 (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Ct32b2 (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Ct32b3 (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Ct32b4 (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Dmac0 (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Dmac1 (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Dmic0 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Flexcomm0 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Flexcomm1 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Flexcomm2 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Flexcomm3 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Flexcomm4 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Flexcomm5 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Flexcomm6 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Flexcomm7 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Flexcomm14 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Flexcomm15 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Flexspi (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Freqme (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_HashCrypt (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_HsGpio0 (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_HsGpio1 (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_HsGpio2 (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_HsGpio3 (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_HsGpio4 (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_HsGpio5 (*C enumerator*), 230

drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_HsGpio6 (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_HsGpio7 (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2c0 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2c1 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2c2 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2c3 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2c4 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2c5 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2c6 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2c7 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2c15 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2s0 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2s1 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2s2 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2s3 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2s4 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2s5 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2s6 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I2s7 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_I3c0 (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_InputMux (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_IpInvalid (*C enumerator*), 225
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Mrt0 (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Mu (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_OsEventTimer (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_OtpCtrl (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Pint (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_PowerQuad (*C enumerator*), 225
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Puf (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Rng (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_RomCtrlr (*C enumerator*), 225
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Rtc (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Sct (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Sdio0 (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Sdio1 (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Sema (*C enumerator*), 230
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_ShsGpio0 (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Spi0 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Spi1 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Spi2 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Spi3 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Spi4 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Spi5 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Spi6 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Spi7 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Spi14 (*C enumerator*), 229
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Usart0 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Usart1 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Usart2 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Usart3 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Usart4 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Usart5 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Usart6 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Usart7 (*C enumerator*), 228
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_UsbhsDevice (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_UsbhsHost (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_UsbhsPhy (*C enumerator*), 226
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_UsbhsSram (*C enumerator*), 226

drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Utick0 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Wwdt0 (*C enumerator*), 227
drivers_RT_RT600_MIMXRT685S._clock_ip_name.kCLOCK_Wwdt1 (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_name (*C enum*), 231
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_AcmpClk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_AdcClk (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_BusClk (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_ClockOutClk (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_CoreSysClk (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_DmicClk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_DspCpuClk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Flexcomm0Clk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Flexcomm1Clk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Flexcomm2Clk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Flexcomm3Clk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Flexcomm4Clk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Flexcomm5Clk (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Flexcomm6Clk (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Flexcomm7Clk (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Flexcomm14Clk (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Flexcomm15Clk (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_FlexspiClk (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_I3cClk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_MclkClk (*C enumerator*), 231
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_SctClk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Sdio0Clk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Sdio1Clk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_SystickClk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_UsbClk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Wdt0Clk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_name.kCLOCK_Wdt1Clk (*C enumerator*), 232
drivers_RT_RT600_MIMXRT685S._clock_pfd (*C enum*), 233
drivers_RT_RT600_MIMXRT685S._clock_pfd.kCLOCK_Pfd0 (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_pfd.kCLOCK_Pfd1 (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_pfd.kCLOCK_Pfd2 (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_pfd.kCLOCK_Pfd3 (*C enumerator*), 233
drivers_RT_RT600_MIMXRT685S._clock_sys_pll_config (*C struct*), 263
drivers_RT_RT600_MIMXRT685S._compensation (*C enum*), 305
drivers_RT_RT600_MIMXRT685S._compensation.kDMIC_CompValueNegativePoint13 (*C enumerator*), 306
drivers_RT_RT600_MIMXRT685S._compensation.kDMIC_CompValueNegativePoint15 (*C enumerator*), 306
drivers_RT_RT600_MIMXRT685S._compensation.kDMIC_CompValueNegativePoint16 (*C enumerator*), 306
drivers_RT_RT600_MIMXRT685S._compensation.kDMIC_CompValueZero (*C enumerator*), 306
drivers_RT_RT600_MIMXRT685S._crc_config (*C struct*), 266
drivers_RT_RT600_MIMXRT685S._crc_config.complementIn (*C var*), 266
drivers_RT_RT600_MIMXRT685S._crc_config.complementOut (*C var*), 266
drivers_RT_RT600_MIMXRT685S._crc_config.polynomial (*C var*), 266
drivers_RT_RT600_MIMXRT685S._crc_config.reverseIn (*C var*), 266
drivers_RT_RT600_MIMXRT685S._crc_config.reverseOut (*C var*), 266
drivers_RT_RT600_MIMXRT685S._crc_config.seed (*C var*), 266
drivers_RT_RT600_MIMXRT685S._crc_polynomial (*C enum*), 264
drivers_RT_RT600_MIMXRT685S._crc_polynomial.kCRC_Polynomial_CRC_16 (*C enumerator*), 264
drivers_RT_RT600_MIMXRT685S._crc_polynomial.kCRC_Polynomial_CRC_32 (*C enumerator*), 264

drivers_RT_RT600_MIMXRT685S._crc_polynomial.kCRC_Polynomial_CRC_CCITT (*C enumerator*), 264
drivers_RT_RT600_MIMXRT685S._ctimer_capture_channel (*C enum*), 269
drivers_RT_RT600_MIMXRT685S._ctimer_capture_channel.kCTIMER_Capture_0 (*C enumerator*), 269
drivers_RT_RT600_MIMXRT685S._ctimer_capture_channel.kCTIMER_Capture_1 (*C enumerator*), 269
drivers_RT_RT600_MIMXRT685S._ctimer_capture_channel.kCTIMER_Capture_3 (*C enumerator*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_capture_edge (*C enum*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_capture_edge.kCTIMER_Capture_BothEdge (*C enumerator*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_capture_edge.kCTIMER_Capture_FallEdge (*C enumerator*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_capture_edge.kCTIMER_Capture_RiseEdge (*C enumerator*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_config (*C struct*), 276
drivers_RT_RT600_MIMXRT685S._ctimer_config.input (*C var*), 276
drivers_RT_RT600_MIMXRT685S._ctimer_config.mode (*C var*), 276
drivers_RT_RT600_MIMXRT685S._ctimer_config.prescale (*C var*), 277
drivers_RT_RT600_MIMXRT685S._ctimer_external_match (*C enum*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_external_match.kCTIMER_External_Match_0 (*C enumerator*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_external_match.kCTIMER_External_Match_1 (*C enumerator*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_external_match.kCTIMER_External_Match_2 (*C enumerator*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_external_match.kCTIMER_External_Match_3 (*C enumerator*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_interrupt_enable (*C enum*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_interrupt_enable.kCTIMER_Match0InterruptEnable (*C enumerator*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_interrupt_enable.kCTIMER_Match1InterruptEnable (*C enumerator*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_interrupt_enable.kCTIMER_Match2InterruptEnable (*C enumerator*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_interrupt_enable.kCTIMER_Match3InterruptEnable (*C enumerator*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_match (*C enum*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_match_config (*C struct*), 276
drivers_RT_RT600_MIMXRT685S._ctimer_match_config.enableCounterReset (*C var*), 276
drivers_RT_RT600_MIMXRT685S._ctimer_match_config.enableCounterStop (*C var*), 276
drivers_RT_RT600_MIMXRT685S._ctimer_match_config.enableInterrupt (*C var*), 276
drivers_RT_RT600_MIMXRT685S._ctimer_match_config.matchValue (*C var*), 276
drivers_RT_RT600_MIMXRT685S._ctimer_match_config.outControl (*C var*), 276
drivers_RT_RT600_MIMXRT685S._ctimer_match_config.outPinInitState (*C var*), 276
drivers_RT_RT600_MIMXRT685S._ctimer_match_output_control (*C enum*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_match_output_control.kCTIMER_Output_Clear (*C enumerator*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_match_output_control.kCTIMER_Output_NoAction (*C enumerator*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_match_output_control.kCTIMER_Output_Set (*C enumerator*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_match_output_control.kCTIMER_Output_Toggle (*C enumerator*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_match.kCTIMER_Match_0 (*C enumerator*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_match.kCTIMER_Match_1 (*C enumerator*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_match.kCTIMER_Match_2 (*C enumerator*), 270

drivers_RT_RT600_MIMXRT685S._ctimer_match.kCTIMER_Match_3 (*C enumerator*), 270
drivers_RT_RT600_MIMXRT685S._ctimer_status_flags (*C enum*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_status_flags.kCTIMER_Match0Flag (*C enumerator*), 272
drivers_RT_RT600_MIMXRT685S._ctimer_status_flags.kCTIMER_Match1Flag (*C enumerator*), 272
drivers_RT_RT600_MIMXRT685S._ctimer_status_flags.kCTIMER_Match2Flag (*C enumerator*), 272
drivers_RT_RT600_MIMXRT685S._ctimer_status_flags.kCTIMER_Match3Flag (*C enumerator*), 272
drivers_RT_RT600_MIMXRT685S._ctimer_timer_mode (*C enum*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_timer_mode.kCTIMER_IncreaseOnBothEdge (*C enumerator*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_timer_mode.kCTIMER_IncreaseOnFallEdge (*C enumerator*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_timer_mode.kCTIMER_IncreaseOnRiseEdge (*C enumerator*), 271
drivers_RT_RT600_MIMXRT685S._ctimer_timer_mode.kCTIMER_TimerMode (*C enumerator*), 271
drivers_RT_RT600_MIMXRT685S._dc_removal (*C enum*), 306
drivers_RT_RT600_MIMXRT685S._dc_removal.kDMIC_DcCut39 (*C enumerator*), 306
drivers_RT_RT600_MIMXRT685S._dc_removal.kDMIC_DcCut78 (*C enumerator*), 306
drivers_RT_RT600_MIMXRT685S._dc_removal.kDMIC_DcCut155 (*C enumerator*), 306
drivers_RT_RT600_MIMXRT685S._dc_removal.kDMIC_DcNoRemove (*C enumerator*), 306
drivers_RT_RT600_MIMXRT685S._dma_burst_wrap (*C enum*), 290
drivers_RT_RT600_MIMXRT685S._dma_burst_wrap.kDMA_DstWrap (*C enumerator*), 290
drivers_RT_RT600_MIMXRT685S._dma_burst_wrap.kDMA_NoWrap (*C enumerator*), 290
drivers_RT_RT600_MIMXRT685S._dma_burst_wrap.kDMA_SrcAndDstWrap (*C enumerator*), 290
drivers_RT_RT600_MIMXRT685S._dma_burst_wrap.kDMA_SrcWrap (*C enumerator*), 290
drivers_RT_RT600_MIMXRT685S._dma_channel_config (*C struct*), 294
drivers_RT_RT600_MIMXRT685S._dma_channel_config.dstStartAddr (*C var*), 294
drivers_RT_RT600_MIMXRT685S._dma_channel_config.isPeriph (*C var*), 294
drivers_RT_RT600_MIMXRT685S._dma_channel_config.nextDesc (*C var*), 294
drivers_RT_RT600_MIMXRT685S._dma_channel_config.srcStartAddr (*C var*), 294
drivers_RT_RT600_MIMXRT685S._dma_channel_config.trigger (*C var*), 294
drivers_RT_RT600_MIMXRT685S._dma_channel_config.xferCfg (*C var*), 294
drivers_RT_RT600_MIMXRT685S._dma_channel_trigger (*C struct*), 294
drivers_RT_RT600_MIMXRT685S._dma_channel_trigger.burst (*C var*), 294
drivers_RT_RT600_MIMXRT685S._dma_channel_trigger.type (*C var*), 294
drivers_RT_RT600_MIMXRT685S._dma_channel_trigger.wrap (*C var*), 294
drivers_RT_RT600_MIMXRT685S._dma_descriptor (*C struct*), 293
drivers_RT_RT600_MIMXRT685S._dma_descriptor.dstEndAddr (*C var*), 293
drivers_RT_RT600_MIMXRT685S._dma_descriptor.linkToNextDesc (*C var*), 293
drivers_RT_RT600_MIMXRT685S._dma_descriptor.srcEndAddr (*C var*), 293
drivers_RT_RT600_MIMXRT685S._dma_descriptor.xfercfg (*C var*), 293
drivers_RT_RT600_MIMXRT685S._dma_handle (*C struct*), 295
drivers_RT_RT600_MIMXRT685S._dma_handle.base (*C var*), 295
drivers_RT_RT600_MIMXRT685S._dma_handle.callback (*C var*), 295
drivers_RT_RT600_MIMXRT685S._dma_handle.channel (*C var*), 295
drivers_RT_RT600_MIMXRT685S._dma_handle.userData (*C var*), 295
drivers_RT_RT600_MIMXRT685S._dma_int (*C enum*), 287
drivers_RT_RT600_MIMXRT685S._dma_int.kDMA_IntA (*C enumerator*), 288
drivers_RT_RT600_MIMXRT685S._dma_int.kDMA_IntB (*C enumerator*), 288
drivers_RT_RT600_MIMXRT685S._dma_int.kDMA_IntError (*C enumerator*), 288
drivers_RT_RT600_MIMXRT685S._dma_priority (*C enum*), 287
drivers_RT_RT600_MIMXRT685S._dma_priority.kDMA_ChannelPriority0 (*C enumerator*), 287
drivers_RT_RT600_MIMXRT685S._dma_priority.kDMA_ChannelPriority1 (*C enumerator*), 287
drivers_RT_RT600_MIMXRT685S._dma_priority.kDMA_ChannelPriority2 (*C enumerator*), 287

drivers_RT_RT600_MIMXRT685S._dma_priority.kDMA_ChannelPriority3 (*C enumerator*), 287
drivers_RT_RT600_MIMXRT685S._dma_priority.kDMA_ChannelPriority4 (*C enumerator*), 287
drivers_RT_RT600_MIMXRT685S._dma_priority.kDMA_ChannelPriority5 (*C enumerator*), 287
drivers_RT_RT600_MIMXRT685S._dma_priority.kDMA_ChannelPriority6 (*C enumerator*), 287
drivers_RT_RT600_MIMXRT685S._dma_priority.kDMA_ChannelPriority7 (*C enumerator*), 287
drivers_RT_RT600_MIMXRT685S._dma_transfer_config (*C struct*), 294
drivers_RT_RT600_MIMXRT685S._dma_transfer_config.dstAddr (*C var*), 295
drivers_RT_RT600_MIMXRT685S._dma_transfer_config.isPeriph (*C var*), 295
drivers_RT_RT600_MIMXRT685S._dma_transfer_config.nextDesc (*C var*), 295
drivers_RT_RT600_MIMXRT685S._dma_transfer_config.srcAddr (*C var*), 295
drivers_RT_RT600_MIMXRT685S._dma_transfer_config.xfercfg (*C var*), 295
drivers_RT_RT600_MIMXRT685S._dma_transfer_type (*C enum*), 290
drivers_RT_RT600_MIMXRT685S._dma_transfer_type.kDMA_MemoryToMemory (*C enumerator*), 290
drivers_RT_RT600_MIMXRT685S._dma_transfer_type.kDMA_MemoryToPeripheral (*C enumerator*), 290
drivers_RT_RT600_MIMXRT685S._dma_transfer_type.kDMA_PeripheralToMemory (*C enumerator*), 290
drivers_RT_RT600_MIMXRT685S._dma_transfer_type.kDMA_StaticToStatic (*C enumerator*), 290
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst (*C enum*), 289
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_EdgeBurstTransfer1 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_EdgeBurstTransfer2 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_EdgeBurstTransfer4 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_EdgeBurstTransfer8 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_EdgeBurstTransfer16 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_EdgeBurstTransfer32 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_EdgeBurstTransfer64 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_EdgeBurstTransfer128 (*C enumerator*), 290
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_EdgeBurstTransfer256 (*C enumerator*), 290
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_EdgeBurstTransfer512 (*C enumerator*), 290
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_EdgeBurstTransfer1024 (*C enumerator*), 290
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_LevelBurstTransfer (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S._dma_trigger_burst.kDMA_SingleTransfer (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S._dma_trigger_type (*C enum*), 288
drivers_RT_RT600_MIMXRT685S._dma_trigger_type.kDMA_FallingEdgeTrigger (*C enumerator*), 288
drivers_RT_RT600_MIMXRT685S._dma_trigger_type.kDMA_HighLevelTrigger (*C enumerator*), 288
drivers_RT_RT600_MIMXRT685S._dma_trigger_type.kDMA_LowLevelTrigger (*C enumerator*), 288
drivers_RT_RT600_MIMXRT685S._dma_trigger_type.kDMA_NoTrigger (*C enumerator*), 288
drivers_RT_RT600_MIMXRT685S._dma_trigger_type.kDMA_RisingEdgeTrigger (*C enumerator*), 288
drivers_RT_RT600_MIMXRT685S._dma_xfercfg (*C struct*), 293
drivers_RT_RT600_MIMXRT685S._dma_xfercfg.byteWidth (*C var*), 293
drivers_RT_RT600_MIMXRT685S._dma_xfercfg.clrtrig (*C var*), 293

drivers_RT_RT600_MIMXRT685S._dma_xfercfg.dstInc (*C var*), 294
drivers_RT_RT600_MIMXRT685S._dma_xfercfg.intA (*C var*), 293
drivers_RT_RT600_MIMXRT685S._dma_xfercfg.intB (*C var*), 293
drivers_RT_RT600_MIMXRT685S._dma_xfercfg.reload (*C var*), 293
drivers_RT_RT600_MIMXRT685S._dma_xfercfg.srcInc (*C var*), 293
drivers_RT_RT600_MIMXRT685S._dma_xfercfg.swtrig (*C var*), 293
drivers_RT_RT600_MIMXRT685S._dma_xfercfg.transferCount (*C var*), 294
drivers_RT_RT600_MIMXRT685S._dma_xfercfg.valid (*C var*), 293
drivers_RT_RT600_MIMXRT685S._dmic_channel (*C enum*), 306
drivers_RT_RT600_MIMXRT685S._dmic_channel_config (*C struct*), 307
drivers_RT_RT600_MIMXRT685S._dmic_channel_config.dc_cut_level (*C var*), 308
drivers_RT_RT600_MIMXRT685S._dmic_channel_config.divhfclk (*C var*), 308
drivers_RT_RT600_MIMXRT685S._dmic_channel_config.enableSignExtend (*C var*), 308
drivers_RT_RT600_MIMXRT685S._dmic_channel_config.gainshft (*C var*), 308
drivers_RT_RT600_MIMXRT685S._dmic_channel_config.osr (*C var*), 308
drivers_RT_RT600_MIMXRT685S._dmic_channel_config.post_dc_gain_reduce (*C var*), 308
drivers_RT_RT600_MIMXRT685S._dmic_channel_config.preac2coef (*C var*), 308
drivers_RT_RT600_MIMXRT685S._dmic_channel_config.preac4coef (*C var*), 308
drivers_RT_RT600_MIMXRT685S._dmic_channel_config.sample_rate (*C var*), 308
drivers_RT_RT600_MIMXRT685S._dmic_channel_config.saturate16bit (*C var*), 308
drivers_RT_RT600_MIMXRT685S._dmic_channel.kDMIC_Channel0 (*C enumerator*), 306
drivers_RT_RT600_MIMXRT685S._dmic_channel.kDMIC_Channel1 (*C enumerator*), 306
drivers_RT_RT600_MIMXRT685S._dmic_channel.kDMIC_ChannelMAX (*C enumerator*), 306
drivers_RT_RT600_MIMXRT685S._dmic_dma_handle (*C struct*), 297
drivers_RT_RT600_MIMXRT685S._dmic_dma_handle.base (*C var*), 297
drivers_RT_RT600_MIMXRT685S._dmic_dma_handle.callback (*C var*), 297
drivers_RT_RT600_MIMXRT685S._dmic_dma_handle.channel (*C var*), 298
drivers_RT_RT600_MIMXRT685S._dmic_dma_handle.desLink (*C var*), 298
drivers_RT_RT600_MIMXRT685S._dmic_dma_handle.isChannelValid (*C var*), 298
drivers_RT_RT600_MIMXRT685S._dmic_dma_handle.linkNum (*C var*), 298
drivers_RT_RT600_MIMXRT685S._dmic_dma_handle.rxDmaHandle (*C var*), 297
drivers_RT_RT600_MIMXRT685S._dmic_dma_handle.state (*C var*), 298
drivers_RT_RT600_MIMXRT685S._dmic_dma_handle.transferSize (*C var*), 298
drivers_RT_RT600_MIMXRT685S._dmic_dma_handle.userData (*C var*), 298
drivers_RT_RT600_MIMXRT685S._dmic_phy_sample_rate (*C enum*), 307
drivers_RT_RT600_MIMXRT685S._dmic_phy_sample_rate.kDMIC_PhyFullSpeed (*C enumerator*), 307
drivers_RT_RT600_MIMXRT685S._dmic_phy_sample_rate.kDMIC_PhyHalfSpeed (*C enumerator*), 307
drivers_RT_RT600_MIMXRT685S._dmic_transfer (*C struct*), 297
drivers_RT_RT600_MIMXRT685S._dmic_transfer.data (*C var*), 297
drivers_RT_RT600_MIMXRT685S._dmic_transfer.dataAddrInterleaveSize (*C var*), 297
drivers_RT_RT600_MIMXRT685S._dmic_transfer.dataSize (*C var*), 297
drivers_RT_RT600_MIMXRT685S._dmic_transfer.dataWidth (*C var*), 297
drivers_RT_RT600_MIMXRT685S._dmic_transfer.linkTransfer (*C var*), 297
drivers_RT_RT600_MIMXRT685S._dsp_copy_image (*C struct*), 309
drivers_RT_RT600_MIMXRT685S._flexspi_addr_map_config (*C struct*), 328
drivers_RT_RT600_MIMXRT685S._flexspi_addr_map_config.addrEnd (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_addr_map_config.addrOffset (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_addr_map_config.addrStart (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_addr_map_config.remapEnable (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_error_code (*C enum*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_error_code.kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_error_code.kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_error_code.kFLEXSPI_AhbCmdErrorNoError (*C enumerator*), 322

drivers_RT_RT600_MIMXRT685S._flexspi_ahb_error_code.kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_error_code.kFLEXSPI_AhbCmdErrorUnknownOpCode (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_error_code.kFLEXSPI_AhbCmdSequenceExecutionTimeout (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_write_wait_unit (*C enum*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit2AhbCycle (*C enumerator*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit8AhbCycle (*C enumerator*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit32AhbCycle (*C enumerator*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit128AhbCycle (*C enumerator*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit512AhbCycle (*C enumerator*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit2048AhbCycle (*C enumerator*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit8192AhbCycle (*C enumerator*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_ahb_write_wait_unit.kFLEXSPI_AhbWriteWaitUnit32768AhbCycle (*C enumerator*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_ahbBuffer_config (*C struct*), 325
drivers_RT_RT600_MIMXRT685S._flexspi_ahbBuffer_config.bufferSize (*C var*), 325
drivers_RT_RT600_MIMXRT685S._flexspi_ahbBuffer_config.enablePrefetch (*C var*), 325
drivers_RT_RT600_MIMXRT685S._flexspi_ahbBuffer_config.masterIndex (*C var*), 325
drivers_RT_RT600_MIMXRT685S._flexspi_ahbBuffer_config.priority (*C var*), 325
drivers_RT_RT600_MIMXRT685S._flexspi_ahbBuffers_ctrl (*C struct*), 325
drivers_RT_RT600_MIMXRT685S._flexspi_ahbBuffers_ctrl.buffer (*C var*), 325
drivers_RT_RT600_MIMXRT685S._flexspi_arb_command_source (*C enum*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_arb_command_source.kFLEXSPI_AhbReadCommand (*C enumerator*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_arb_command_source.kFLEXSPI_AhbWriteCommand (*C enumerator*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_arb_command_source.kFLEXSPI_IpCommand (*C enumerator*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_arb_command_source.kFLEXSPI_SuspendedCommand (*C enumerator*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_command_type (*C enum*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_command_type.kFLEXSPI_Command (*C enumerator*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_command_type.kFLEXSPI_Config (*C enumerator*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_command_type.kFLEXSPI_Read (*C enumerator*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_command_type.kFLEXSPI_Write (*C enumerator*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_config (*C struct*), 325
drivers_RT_RT600_MIMXRT685S._flexspi_config.ahbConfig (*C struct*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_config.ahbConfig.ahbBusTimeoutCycle (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_config.ahbConfig.ahbGrantTimeoutCycle (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_config.ahbConfig.buffer (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_config.ahbConfig.disableAhbReadResume (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_config.ahbConfig.enableAHBBufferable (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_config.ahbConfig.enableAHBCachable (*C var*), 330
drivers_RT_RT600_MIMXRT685S._flexspi_config.ahbConfig.enableAHBPrefetch (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_config.ahbConfig.enableClearAHBBufferOpt (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_config.ahbConfig.enableReadAddressOpt (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_config.ahbConfig.resumeWaitCycle (*C var*), 329
drivers_RT_RT600_MIMXRT685S._flexspi_config.clockDiv (*C var*), 325

drivers_RT_RT600_MIMXRT685S._flexspi_config.enableDoze (*C var*), 325
drivers_RT_RT600_MIMXRT685S._flexspi_config.enableHalfSpeedAccess (*C var*), 325
drivers_RT_RT600_MIMXRT685S._flexspi_config.enableSameConfigForAll (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_config.enableSckFreeRunning (*C var*), 325
drivers_RT_RT600_MIMXRT685S._flexspi_config.ipGrantTimeoutCycle (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_config.rxSampleClock (*C var*), 325
drivers_RT_RT600_MIMXRT685S._flexspi_config.rxSampleClockDiff (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_config.rxSampleClockPortB (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_config.rxWatermark (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_config.seqTimeoutCycle (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_config.txWatermark (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_cs_interval_cycle_unit (*C enum*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_cs_interval_cycle_unit.kFLEXSPI_CsIntervalUnit1SckCycle (*C enumerator*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_cs_interval_cycle_unit.kFLEXSPI_CsIntervalUnit256SckCycle (*C enumerator*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_device_config (*C struct*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.addressShift (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.AHBWriteWaitInterval (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.AHBWriteWaitUnit (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.ARDSeqIndex (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.ARDSeqNumber (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.AWRSeqIndex (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.AWRSeqNumber (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.columnspace (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.CSHoldTime (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.CSInterval (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.CSIntervalUnit (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.CSSetupTime (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.dataValidTime (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.enableWordAddress (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.enableWriteMask (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.flashSize (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.flexspiRootClk (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.isFroClockSource (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_device_config.isSck2Enabled (*C var*), 326
drivers_RT_RT600_MIMXRT685S._flexspi_dma_handle (*C struct*), 332
drivers_RT_RT600_MIMXRT685S._flexspi_dma_handle.completionCallback (*C var*), 332
drivers_RT_RT600_MIMXRT685S._flexspi_dma_handle.count (*C var*), 332
drivers_RT_RT600_MIMXRT685S._flexspi_dma_handle.nbytes (*C var*), 332
drivers_RT_RT600_MIMXRT685S._flexspi_dma_handle.nsize (*C var*), 332
drivers_RT_RT600_MIMXRT685S._flexspi_dma_handle.rxDmaHandle (*C var*), 332
drivers_RT_RT600_MIMXRT685S._flexspi_dma_handle.state (*C var*), 332
drivers_RT_RT600_MIMXRT685S._flexspi_dma_handle.transferSize (*C var*), 332
drivers_RT_RT600_MIMXRT685S._flexspi_dma_handle.txDmaHandle (*C var*), 332
drivers_RT_RT600_MIMXRT685S._flexspi_dma_handle.userData (*C var*), 332
drivers_RT_RT600_MIMXRT685S._flexspi_dma_ntransfer_size (*C enum*), 331
drivers_RT_RT600_MIMXRT685S._flexspi_dma_ntransfer_size.kFLEXPSI_DMAnSize1Bytes (*C enumerator*), 331
drivers_RT_RT600_MIMXRT685S._flexspi_dma_ntransfer_size.kFLEXPSI_DMAnSize2Bytes (*C enumerator*), 331
drivers_RT_RT600_MIMXRT685S._flexspi_dma_ntransfer_size.kFLEXPSI_DMAnSize4Bytes (*C enumerator*), 331
drivers_RT_RT600_MIMXRT685S._flexspi_flags (*C enum*), 319
drivers_RT_RT600_MIMXRT685S._flexspi_flags.kFLEXSPI_AhbBusErrorFlag (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_flags.kFLEXSPI_AhbCommandGrantTimeoutFlag (*C enumerator*), 320

drivers_RT_RT600_MIMXRT685S._flexspi_flags.kFLEXSPI_AhbCommandSequenceErrorFlag (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_flags.kFLEXSPI_AllInterruptFlags (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_flags.kFLEXSPI_IpCommandExecutionDoneFlag (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_flags.kFLEXSPI_IpCommandGrantTimeoutFlag (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_flags.kFLEXSPI_IpCommandSequenceErrorFlag (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_flags.kFLEXSPI_IpRxFifoWatermarkAvailableFlag (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_flags.kFLEXSPI_IpTxFifoWatermarkEmptyFlag (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_flags.kFLEXSPI_SckStoppedBecauseRxFullFlag (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_flags.kFLEXSPI_SckStoppedBecauseTxEmptyFlag (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_flags.kFLEXSPI_SequenceExecutionTimeoutFlag (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_handle (*C struct*), 328
drivers_RT_RT600_MIMXRT685S._flexspi_handle.completionCallback (*C var*), 328
drivers_RT_RT600_MIMXRT685S._flexspi_handle.data (*C var*), 328
drivers_RT_RT600_MIMXRT685S._flexspi_handle.dataSize (*C var*), 328
drivers_RT_RT600_MIMXRT685S._flexspi_handle.state (*C var*), 328
drivers_RT_RT600_MIMXRT685S._flexspi_handle.transferTotalSize (*C var*), 328
drivers_RT_RT600_MIMXRT685S._flexspi_handle.userData (*C var*), 328
drivers_RT_RT600_MIMXRT685S._flexspi_ip_error_code (*C enum*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorInvalidAddress (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorNoError (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorSequenceExecutionTimeout (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_ip_error_code.kFLEXSPI_IpCmdErrorUnknownOpCode (*C enumerator*), 322
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config (*C struct*), 458
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.blockSize (*C var*), 458
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.flashStateCtx (*C var*), 459
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.halfClkForNonReadCmd (*C var*), 458
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.ipcmdSerialClkFreq (*C var*), 458
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.isDataOrderSwapped (*C var*), 458
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.isUniformBlockSize (*C var*), 458
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.memConfig (*C var*), 458
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.needExitNoCmdMode (*C var*), 458
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.needRestoreNoCmdMode (*C var*), 458
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.pageSize (*C var*), 458
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.reserve2 (*C var*), 459
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.reserved0 (*C var*), 458
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.sectorSize (*C var*), 458
drivers_RT_RT600_MIMXRT685S._flexspi_nor_config.serialNorType (*C var*), 458

drivers_RT_RT600_MIMXRT685S._flexspi_pad (*C enum*), 319
drivers_RT_RT600_MIMXRT685S._flexspi_pad.kFLEXSPI_1PAD (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S._flexspi_pad.kFLEXSPI_2PAD (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S._flexspi_pad.kFLEXSPI_4PAD (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S._flexspi_pad.kFLEXSPI_8PAD (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S._flexspi_port (*C enum*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_port.kFLEXSPI_PortA1 (*C enumerator*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_port.kFLEXSPI_PortA2 (*C enumerator*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_port.kFLEXSPI_PortCount (*C enumerator*), 323
drivers_RT_RT600_MIMXRT685S._flexspi_read_sample_clock (*C enum*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_read_sample_clock.kFLEXSPI_ReadSampleClkExternalInputFromDqsP (*C enumerator*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_read_sample_clock.kFLEXSPI_ReadSampleClkLoopbackFromDqsPad (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_read_sample_clock.kFLEXSPI_ReadSampleClkLoopbackFromSckPad (*C enumerator*), 321
drivers_RT_RT600_MIMXRT685S._flexspi_read_sample_clock.kFLEXSPI_ReadSampleClkLoopbackInternally (*C enumerator*), 320
drivers_RT_RT600_MIMXRT685S._flexspi_status (*C enum*), 444
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPI_DeviceTimeout (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPI_Fail (*C enumerator*), 444
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPI_InvalidArgument (*C enumerator*), 444
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPI_InvalidSequence (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPI_SequenceExecutionTimeout (*C enumerator*), 444
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPI_Success (*C enumerator*), 444
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPINOR_CommandFailure (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPINOR_EraseAllFail (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPINOR_EraseSectorFail (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPINOR_Flash_NotFound (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPINOR_NotSupported (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPINOR_ProgramFail (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPINOR_SFDP_NotFound (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPINOR_Unsupported_SFDP_Version (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPINOR_WaitTimeout (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_status.kStatus_FLEXSPINOR_WriteAlignmentError (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S._flexspi_transfer (*C struct*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_transfer.cmdType (*C var*), 328
drivers_RT_RT600_MIMXRT685S._flexspi_transfer.data (*C var*), 328
drivers_RT_RT600_MIMXRT685S._flexspi_transfer.dataSize (*C var*), 328
drivers_RT_RT600_MIMXRT685S._flexspi_transfer.deviceAddress (*C var*), 327
drivers_RT_RT600_MIMXRT685S._flexspi_transfer.port (*C var*), 328
drivers_RT_RT600_MIMXRT685S._flexspi_transfer.seqIndex (*C var*), 328

drivers_RT_RT600_MIMXRT685S._flexspi_transfer.SeqNumber (*C var*), 328
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig (*C struct*), 454
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.busyBitPolarity (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.busyOffset (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.columnAddressWidth (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.commandInterval (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.configCmdArgs (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.configCmdEnable (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.configCmdSeqs (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.configModeType (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.controllerMiscOption (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.csHoldTime (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.csPadSettingOverride (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.csSetupTime (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.dataPadSettingOverride (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.dataValidTime (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.deviceModeArg (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.deviceModeCfgEnable (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.deviceModeSeq (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.deviceModeType (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.deviceType (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.dqsPadSettingOverride (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.lookupTable (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.lutCustomSeq (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.lutCustomSeqEnable (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.readSampleClkSrc (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.reserved0 (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.reserved1 (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.reserved2 (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.reserved3 (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.reserved4 (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.sclkPadSettingOverride (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.serialClkFreq (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.sflashA1Size (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.sflashA2Size (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.sflashB1Size (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.sflashB2Size (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.sflashPadType (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.tag (*C var*), 454
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.timeoutInMs (*C var*), 456
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.version (*C var*), 454
drivers_RT_RT600_MIMXRT685S._FlexSPIConfig.waitTimeCfgCommands (*C var*), 455
drivers_RT_RT600_MIMXRT685S._FlexSPIOperationType (*C enum*), 449
drivers_RT_RT600_MIMXRT685S._FlexSPIOperationType.kFlexSpiOperation_Command (*C enumerator*), 449
drivers_RT_RT600_MIMXRT685S._FlexSPIOperationType.kFlexSpiOperation_Config (*C enumerator*), 449
drivers_RT_RT600_MIMXRT685S._FlexSPIOperationType.kFlexSpiOperation_End (*C enumerator*), 449
drivers_RT_RT600_MIMXRT685S._FlexSPIOperationType.kFlexSpiOperation_Read (*C enumerator*), 449
drivers_RT_RT600_MIMXRT685S._FlexSPIOperationType.kFlexSpiOperation_Write (*C enumerator*), 449
drivers_RT_RT600_MIMXRT685S._FlexSpiXfer (*C struct*), 457
drivers_RT_RT600_MIMXRT685S._FlexSpiXfer.baseAddress (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSpiXfer.isParallelModeEnable (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSpiXfer.operation (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSpiXfer.rxBuffer (*C var*), 458

drivers_RT_RT600_MIMXRT685S._FlexSpiXfer.rxSize (*C var*), 458
drivers_RT_RT600_MIMXRT685S._FlexSpiXfer.seqId (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSpiXfer.seqNum (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSpiXfer.txBuffer (*C var*), 457
drivers_RT_RT600_MIMXRT685S._FlexSpiXfer.txSize (*C var*), 457
drivers_RT_RT600_MIMXRT685S._gpio_interrupt_config (*C struct*), 531
drivers_RT_RT600_MIMXRT685S._gpio_interrupt_index (*C enum*), 529
drivers_RT_RT600_MIMXRT685S._gpio_interrupt_index.kGPIO_InterruptA (*C enumerator*), 529
drivers_RT_RT600_MIMXRT685S._gpio_interrupt_index.kGPIO_InterruptB (*C enumerator*), 529
drivers_RT_RT600_MIMXRT685S._gpio_pin_config (*C struct*), 530
drivers_RT_RT600_MIMXRT685S._gpio_pin_config.outputLogic (*C var*), 531
drivers_RT_RT600_MIMXRT685S._gpio_pin_config.pinDirection (*C var*), 531
drivers_RT_RT600_MIMXRT685S._gpio_pin_direction (*C enum*), 529
drivers_RT_RT600_MIMXRT685S._gpio_pin_direction.kGPIO_DigitalInput (*C enumerator*), 529
drivers_RT_RT600_MIMXRT685S._gpio_pin_direction.kGPIO_DigitalOutput (*C enumerator*), 529
drivers_RT_RT600_MIMXRT685S._gpio_pin_enable_mode (*C enum*), 529
drivers_RT_RT600_MIMXRT685S._gpio_pin_enable_mode.kGPIO_PinIntEnableEdge (*C enumerator*), 529
drivers_RT_RT600_MIMXRT685S._gpio_pin_enable_mode.kGPIO_PinIntEnableLevel (*C enumerator*), 529
drivers_RT_RT600_MIMXRT685S._gpio_pin_enable_polarity (*C enum*), 529
drivers_RT_RT600_MIMXRT685S._gpio_pin_enable_polarity.kGPIO_PinIntEnableHighOrRise (*C enumerator*), 529
drivers_RT_RT600_MIMXRT685S._gpio_pin_enable_polarity.kGPIO_PinIntEnableLowOrFall (*C enumerator*), 529
drivers_RT_RT600_MIMXRT685S._hashcrypt_aes_keysize_t (*C enum*), 337
drivers_RT_RT600_MIMXRT685S._hashcrypt_aes_keysize_t.kHASHCRYPT_Aes128 (*C enumerator*), 337
drivers_RT_RT600_MIMXRT685S._hashcrypt_aes_keysize_t.kHASHCRYPT_Aes192 (*C enumerator*), 337
drivers_RT_RT600_MIMXRT685S._hashcrypt_aes_keysize_t.kHASHCRYPT_Aes256 (*C enumerator*), 337
drivers_RT_RT600_MIMXRT685S._hashcrypt_aes_keysize_t.kHASHCRYPT_InvalidKey (*C enumerator*), 337
drivers_RT_RT600_MIMXRT685S._hashcrypt_aes_mode_t (*C enum*), 337
drivers_RT_RT600_MIMXRT685S._hashcrypt_aes_mode_t.kHASHCRYPT_AesCbc (*C enumerator*), 337
drivers_RT_RT600_MIMXRT685S._hashcrypt_aes_mode_t.kHASHCRYPT_AesCtr (*C enumerator*), 337
drivers_RT_RT600_MIMXRT685S._hashcrypt_aes_mode_t.kHASHCRYPT_AesEcb (*C enumerator*), 337
drivers_RT_RT600_MIMXRT685S._hashcrypt_algo_t (*C enum*), 336
drivers_RT_RT600_MIMXRT685S._hashcrypt_algo_t.kHASHCRYPT_Aes (*C enumerator*), 336
drivers_RT_RT600_MIMXRT685S._hashcrypt_algo_t.kHASHCRYPT_Sha1 (*C enumerator*), 336
drivers_RT_RT600_MIMXRT685S._hashcrypt_algo_t.kHASHCRYPT_Sha256 (*C enumerator*), 336
drivers_RT_RT600_MIMXRT685S._hashcrypt_handle (*C struct*), 341
drivers_RT_RT600_MIMXRT685S._hashcrypt_handle.keyType (*C var*), 341
drivers_RT_RT600_MIMXRT685S._hashcrypt_handle.keyWord (*C var*), 341
drivers_RT_RT600_MIMXRT685S._hashcrypt_hash_ctx_t (*C struct*), 343
drivers_RT_RT600_MIMXRT685S._hashcrypt_hash_ctx_t.x (*C var*), 343
drivers_RT_RT600_MIMXRT685S._hashcrypt_key (*C enum*), 337
drivers_RT_RT600_MIMXRT685S._hashcrypt_key.kHASHCRYPT_SecretKey (*C enumerator*), 337
drivers_RT_RT600_MIMXRT685S._hashcrypt_key.kHASHCRYPT_UserKey (*C enumerator*), 337
drivers_RT_RT600_MIMXRT685S._i2c_direction (*C enum*), 356
drivers_RT_RT600_MIMXRT685S._i2c_direction.kI2C_Read (*C enumerator*), 356
drivers_RT_RT600_MIMXRT685S._i2c_direction.kI2C_Write (*C enumerator*), 356
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable (*C enum*), 348

drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_CommonAllInterruptEnable (*C enumerator*), 349
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_EventTimeoutInterruptEnable (*C enumerator*), 349
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_MasterAllInterruptEnable (*C enumerator*), 349
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_MasterArbitrationLostInterruptEnable (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_MasterPendingInterruptEnable (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_MasterStartStopErrorInterruptEnable (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_MonitorIdleInterruptEnable (*C enumerator*), 349
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_MonitorOverflowInterruptEnable (*C enumerator*), 349
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_MonitorReadyInterruptEnable (*C enumerator*), 349
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_SclTimeoutInterruptEnable (*C enumerator*), 349
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_SlaveAllInterruptEnable (*C enumerator*), 349
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_SlaveDeselectedInterruptEnable (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_SlaveNotStretchingInterruptEnable (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_interrupt_enable.kI2C_SlavePendingInterruptEnable (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_master_config (*C struct*), 358
drivers_RT_RT600_MIMXRT685S._i2c_master_config.baudRate_Bps (*C var*), 358
drivers_RT_RT600_MIMXRT685S._i2c_master_config.enableMaster (*C var*), 358
drivers_RT_RT600_MIMXRT685S._i2c_master_config.enableTimeout (*C var*), 358
drivers_RT_RT600_MIMXRT685S._i2c_master_config.timeout_Ms (*C var*), 358
drivers_RT_RT600_MIMXRT685S._i2c_master_dma_handle (*C struct*), 344
drivers_RT_RT600_MIMXRT685S._i2c_master_dma_handle.buf (*C var*), 345
drivers_RT_RT600_MIMXRT685S._i2c_master_dma_handle.checkAddrNack (*C var*), 345
drivers_RT_RT600_MIMXRT685S._i2c_master_dma_handle.completionCallback (*C var*), 345
drivers_RT_RT600_MIMXRT685S._i2c_master_dma_handle.dmaHandle (*C var*), 345
drivers_RT_RT600_MIMXRT685S._i2c_master_dma_handle.remainingBytesDMA (*C var*), 345
drivers_RT_RT600_MIMXRT685S._i2c_master_dma_handle.state (*C var*), 345
drivers_RT_RT600_MIMXRT685S._i2c_master_dma_handle.transfer (*C var*), 345
drivers_RT_RT600_MIMXRT685S._i2c_master_dma_handle.transferCount (*C var*), 345
drivers_RT_RT600_MIMXRT685S._i2c_master_dma_handle.userData (*C var*), 345
drivers_RT_RT600_MIMXRT685S._i2c_master_handle (*C struct*), 359
drivers_RT_RT600_MIMXRT685S._i2c_master_handle.buf (*C var*), 359
drivers_RT_RT600_MIMXRT685S._i2c_master_handle.checkAddrNack (*C var*), 359
drivers_RT_RT600_MIMXRT685S._i2c_master_handle.completionCallback (*C var*), 359
drivers_RT_RT600_MIMXRT685S._i2c_master_handle.remainingBytes (*C var*), 359
drivers_RT_RT600_MIMXRT685S._i2c_master_handle.state (*C var*), 359
drivers_RT_RT600_MIMXRT685S._i2c_master_handle.transfer (*C var*), 359
drivers_RT_RT600_MIMXRT685S._i2c_master_handle.transferCount (*C var*), 359
drivers_RT_RT600_MIMXRT685S._i2c_master_handle.userData (*C var*), 359
drivers_RT_RT600_MIMXRT685S._i2c_master_transfer (*C struct*), 358
drivers_RT_RT600_MIMXRT685S._i2c_master_transfer_flags (*C enum*), 356
drivers_RT_RT600_MIMXRT685S._i2c_master_transfer_flags.kI2C_TransferDefaultFlag (*C enumerator*), 356
drivers_RT_RT600_MIMXRT685S._i2c_master_transfer_flags.kI2C_TransferNoStartFlag (*C enumerator*), 356

drivers_RT_RT600_MIMXRT685S._i2c_master_transfer_flags.kI2C_TransferNoStopFlag (*C enumerator*), 357
drivers_RT_RT600_MIMXRT685S._i2c_master_transfer_flags.kI2C_TransferRepeatedStartFlag (*C enumerator*), 356
drivers_RT_RT600_MIMXRT685S._i2c_master_transfer.data (*C var*), 359
drivers_RT_RT600_MIMXRT685S._i2c_master_transfer.dataSize (*C var*), 359
drivers_RT_RT600_MIMXRT685S._i2c_master_transfer.direction (*C var*), 358
drivers_RT_RT600_MIMXRT685S._i2c_master_transfer.flags (*C var*), 358
drivers_RT_RT600_MIMXRT685S._i2c_master_transfer.slaveAddress (*C var*), 358
drivers_RT_RT600_MIMXRT685S._i2c_master_transfer.subaddress (*C var*), 359
drivers_RT_RT600_MIMXRT685S._i2c_master_transfer.subaddressSize (*C var*), 359
drivers_RT_RT600_MIMXRT685S._i2c_slave_address (*C struct*), 367
drivers_RT_RT600_MIMXRT685S._i2c_slave_address_qual_mode (*C enum*), 365
drivers_RT_RT600_MIMXRT685S._i2c_slave_address_qual_mode.kI2C_QualModeExtend (*C enumerator*), 365
drivers_RT_RT600_MIMXRT685S._i2c_slave_address_qual_mode.kI2C_QualModeMask (*C enumerator*), 365
drivers_RT_RT600_MIMXRT685S._i2c_slave_address_register (*C enum*), 364
drivers_RT_RT600_MIMXRT685S._i2c_slave_address_register.kI2C_SlaveAddressRegister0 (*C enumerator*), 365
drivers_RT_RT600_MIMXRT685S._i2c_slave_address_register.kI2C_SlaveAddressRegister1 (*C enumerator*), 365
drivers_RT_RT600_MIMXRT685S._i2c_slave_address_register.kI2C_SlaveAddressRegister2 (*C enumerator*), 365
drivers_RT_RT600_MIMXRT685S._i2c_slave_address_register.kI2C_SlaveAddressRegister3 (*C enumerator*), 365
drivers_RT_RT600_MIMXRT685S._i2c_slave_address.address (*C var*), 367
drivers_RT_RT600_MIMXRT685S._i2c_slave_address.addressDisable (*C var*), 367
drivers_RT_RT600_MIMXRT685S._i2c_slave_bus_speed (*C enum*), 365
drivers_RT_RT600_MIMXRT685S._i2c_slave_bus_speed.kI2C_SlaveFastMode (*C enumerator*), 365
drivers_RT_RT600_MIMXRT685S._i2c_slave_bus_speed.kI2C_SlaveFastModePlus (*C enumerator*), 365
drivers_RT_RT600_MIMXRT685S._i2c_slave_bus_speed.kI2C_SlaveHsMode (*C enumerator*), 365
drivers_RT_RT600_MIMXRT685S._i2c_slave_bus_speed.kI2C_SlaveStandardMode (*C enumerator*), 365
drivers_RT_RT600_MIMXRT685S._i2c_slave_config (*C struct*), 368
drivers_RT_RT600_MIMXRT685S._i2c_slave_config.address0 (*C var*), 368
drivers_RT_RT600_MIMXRT685S._i2c_slave_config.address1 (*C var*), 368
drivers_RT_RT600_MIMXRT685S._i2c_slave_config.address2 (*C var*), 368
drivers_RT_RT600_MIMXRT685S._i2c_slave_config.address3 (*C var*), 368
drivers_RT_RT600_MIMXRT685S._i2c_slave_config.busSpeed (*C var*), 368
drivers_RT_RT600_MIMXRT685S._i2c_slave_config.enableSlave (*C var*), 368
drivers_RT_RT600_MIMXRT685S._i2c_slave_config.qualAddress (*C var*), 368
drivers_RT_RT600_MIMXRT685S._i2c_slave_config.qualMode (*C var*), 368
drivers_RT_RT600_MIMXRT685S._i2c_slave_fsm (*C enum*), 366
drivers_RT_RT600_MIMXRT685S._i2c_slave_fsm.kI2C_SlaveFsmAddressMatch (*C enumerator*), 366
drivers_RT_RT600_MIMXRT685S._i2c_slave_fsm.kI2C_SlaveFsmReceive (*C enumerator*), 366
drivers_RT_RT600_MIMXRT685S._i2c_slave_fsm.kI2C_SlaveFsmTransmit (*C enumerator*), 366
drivers_RT_RT600_MIMXRT685S._i2c_slave_handle (*C struct*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_handle.callback (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_handle.isBusy (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_handle.slaveFsm (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_handle.transfer (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_handle.userData (*C var*), 370
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer (*C struct*), 368
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer_event (*C enum*), 365

drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer_event.kI2C_SlaveAddressMatchEvent (*C enumerator*), 366
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer_event.kI2C_SlaveAllEvents (*C enumerator*), 366
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer_event.kI2C_SlaveCompletionEvent (*C enumerator*), 366
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer_event.kI2C_SlaveDeselectedEvent (*C enumerator*), 366
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer_event.kI2C_SlaveReceiveEvent (*C enumerator*), 366
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer_event.kI2C_SlaveTransmitEvent (*C enumerator*), 366
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer.completionStatus (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer.event (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer.eventMask (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer.handle (*C var*), 368
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer.receivedAddress (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer.rxData (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer.rxSize (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer.transferredCount (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer.txData (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_slave_transfer.txSize (*C var*), 369
drivers_RT_RT600_MIMXRT685S._i2c_status_flags (*C enum*), 346
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_CommonAllClearFlags (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_EventTimeoutFlag (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MasterAddrNackFlag (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MasterAllClearFlags (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MasterArbitrationLostFlag (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MasterDataNackFlag (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MasterIdleFlag (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MasterPendingFlag (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MasterRxReadyFlag (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MasterStartStopErrorFlag (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MasterTxReadyFlag (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MonitorActiveFlag (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MonitorIdleFlag (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MonitorOverflowFlag (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_MonitorReadyFlag (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SaveDeselected (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SclTimeoutFlag (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SlaveAddress0MatchFlag (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SlaveAddress1MatchFlag (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SlaveAddress2MatchFlag (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SlaveAddress3MatchFlag (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SlaveAddressedFlag (*C enumerator*), 347

drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SlaveAllClearFlags (*C enumerator*), 348
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SlaveNotStretching (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SlavePendingFlag (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SlaveReceiveFlag (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SlaveSelected (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_status_flags.kI2C_SlaveTransmitFlag (*C enumerator*), 347
drivers_RT_RT600_MIMXRT685S._i2c_transfer_states (*C enum*), 357
drivers_RT_RT600_MIMXRT685S._i2c_transfer_states.kIdleState (*C enumerator*), 357
drivers_RT_RT600_MIMXRT685S._i2c_transfer_states.kReceiveDataBeginState (*C enumerator*), 357
drivers_RT_RT600_MIMXRT685S._i2c_transfer_states.kReceiveDataState (*C enumerator*), 357
drivers_RT_RT600_MIMXRT685S._i2c_transfer_states.kReceiveLastDataState (*C enumerator*), 357
drivers_RT_RT600_MIMXRT685S._i2c_transfer_states.kStartState (*C enumerator*), 357
drivers_RT_RT600_MIMXRT685S._i2c_transfer_states.kStopState (*C enumerator*), 357
drivers_RT_RT600_MIMXRT685S._i2c_transfer_states.kTransmitDataState (*C enumerator*), 357
drivers_RT_RT600_MIMXRT685S._i2c_transfer_states.kTransmitSubaddrState (*C enumerator*), 357
drivers_RT_RT600_MIMXRT685S._i2c_transfer_states.kWaitForCompletionState (*C enumerator*), 357
drivers_RT_RT600_MIMXRT685S._i2s_bridge_dataout_mask (*C enum*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_dataout_mask.kI2S_BRIDGE_Flexcomm0DataOut (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_dataout_mask.kI2S_BRIDGE_Flexcomm1DataOut (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_dataout_mask.kI2S_BRIDGE_Flexcomm2DataOut (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_dataout_mask.kI2S_BRIDGE_Flexcomm3DataOut (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_dataout_mask.kI2S_BRIDGE_Flexcomm4DataOut (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_dataout_mask.kI2S_BRIDGE_Flexcomm5DataOut (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_dataout_mask.kI2S_BRIDGE_Flexcomm6DataOut (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_dataout_mask.kI2S_BRIDGE_Flexcomm7DataOut (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_set_index (*C enum*), 370
drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_set_index.kI2S_BRIDGE_OriginalSignal (*C enumerator*), 370
drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_set_index.kI2S_BRIDGE_ShareSet0 (*C enumerator*), 370
drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_set_index.kI2S_BRIDGE_ShareSet1 (*C enumerator*), 370
drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_src (*C enum*), 370
drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_src.kI2S_BRIDGE_Flexcomm0 (*C enumerator*), 370
drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_src.kI2S_BRIDGE_Flexcomm1 (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_src.kI2S_BRIDGE_Flexcomm2 (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_src.kI2S_BRIDGE_Flexcomm3 (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_src.kI2S_BRIDGE_Flexcomm4 (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_src.kI2S_BRIDGE_Flexcomm5 (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_src.kI2S_BRIDGE_Flexcomm6 (*C enumerator*), 371

drivers_RT_RT600_MIMXRT685S._i2s_bridge_share_src.kI2S_BRIDGE_Flexcomm7 (*C enumerator*), 371
drivers_RT_RT600_MIMXRT685S._i2s_bridge_signal (*C enum*), 370
drivers_RT_RT600_MIMXRT685S._i2s_bridge_signal.kI2S_BRIDGE_SignalDataIn (*C enumerator*), 370
drivers_RT_RT600_MIMXRT685S._i2s_bridge_signal.kI2S_BRIDGE_SignalDataOut (*C enumerator*), 370
drivers_RT_RT600_MIMXRT685S._i2s_bridge_signal.kI2S_BRIDGE_SignalSCK (*C enumerator*), 370
drivers_RT_RT600_MIMXRT685S._i2s_bridge_signal.kI2S_BRIDGE_SignalWS (*C enumerator*), 370
drivers_RT_RT600_MIMXRT685S._i2s_config (*C struct*), 384
drivers_RT_RT600_MIMXRT685S._i2s_config.dataLength (*C var*), 384
drivers_RT_RT600_MIMXRT685S._i2s_config.divider (*C var*), 384
drivers_RT_RT600_MIMXRT685S._i2s_config.frameLength (*C var*), 384
drivers_RT_RT600_MIMXRT685S._i2s_config.leftJust (*C var*), 384
drivers_RT_RT600_MIMXRT685S._i2s_config.masterSlave (*C var*), 384
drivers_RT_RT600_MIMXRT685S._i2s_config.mode (*C var*), 384
drivers_RT_RT600_MIMXRT685S._i2s_config.oneChannel (*C var*), 384
drivers_RT_RT600_MIMXRT685S._i2s_config.pack48 (*C var*), 385
drivers_RT_RT600_MIMXRT685S._i2s_config.pdmData (*C var*), 384
drivers_RT_RT600_MIMXRT685S._i2s_config.position (*C var*), 385
drivers_RT_RT600_MIMXRT685S._i2s_config.rightLow (*C var*), 384
drivers_RT_RT600_MIMXRT685S._i2s_config.sckPol (*C var*), 384
drivers_RT_RT600_MIMXRT685S._i2s_config.txEmptyZero (*C var*), 385
drivers_RT_RT600_MIMXRT685S._i2s_config.watermark (*C var*), 385
drivers_RT_RT600_MIMXRT685S._i2s_config.wsPol (*C var*), 384
drivers_RT_RT600_MIMXRT685S._i2s_dma_handle (*C struct*), 375
drivers_RT_RT600_MIMXRT685S._i2s_dma_handle.bytesPerFrame (*C var*), 376
drivers_RT_RT600_MIMXRT685S._i2s_dma_handle.completionCallback (*C var*), 376
drivers_RT_RT600_MIMXRT685S._i2s_dma_handle.dmaHandle (*C var*), 376
drivers_RT_RT600_MIMXRT685S._i2s_dma_handle.i2sLoopDMADescriptor (*C var*), 376
drivers_RT_RT600_MIMXRT685S._i2s_dma_handle.i2sLoopDMADescriptorNum (*C var*), 376
drivers_RT_RT600_MIMXRT685S._i2s_dma_handle.i2sQueue (*C var*), 376
drivers_RT_RT600_MIMXRT685S._i2s_dma_handle.queueDriver (*C var*), 376
drivers_RT_RT600_MIMXRT685S._i2s_dma_handle.queueUser (*C var*), 376
drivers_RT_RT600_MIMXRT685S._i2s_dma_handle.state (*C var*), 376
drivers_RT_RT600_MIMXRT685S._i2s_dma_handle.userData (*C var*), 376
drivers_RT_RT600_MIMXRT685S._i2s_flags (*C enum*), 382
drivers_RT_RT600_MIMXRT685S._i2s_flags.kI2S_RxErrorFlag (*C enumerator*), 382
drivers_RT_RT600_MIMXRT685S._i2s_flags.kI2S_RxLevelFlag (*C enumerator*), 382
drivers_RT_RT600_MIMXRT685S._i2s_flags.kI2S_TxErrorFlag (*C enumerator*), 382
drivers_RT_RT600_MIMXRT685S._i2s_flags.kI2S_TxLevelFlag (*C enumerator*), 382
drivers_RT_RT600_MIMXRT685S._i2s_handle (*C struct*), 385
drivers_RT_RT600_MIMXRT685S._i2s_handle.completionCallback (*C var*), 385
drivers_RT_RT600_MIMXRT685S._i2s_handle.dataLength (*C var*), 385
drivers_RT_RT600_MIMXRT685S._i2s_handle.errorCount (*C var*), 386
drivers_RT_RT600_MIMXRT685S._i2s_handle.i2sQueue (*C var*), 386
drivers_RT_RT600_MIMXRT685S._i2s_handle.oneChannel (*C var*), 385
drivers_RT_RT600_MIMXRT685S._i2s_handle.pack48 (*C var*), 385
drivers_RT_RT600_MIMXRT685S._i2s_handle.queueDriver (*C var*), 386
drivers_RT_RT600_MIMXRT685S._i2s_handle.queueUser (*C var*), 386
drivers_RT_RT600_MIMXRT685S._i2s_handle.state (*C var*), 385
drivers_RT_RT600_MIMXRT685S._i2s_handle.transferCount (*C var*), 386
drivers_RT_RT600_MIMXRT685S._i2s_handle.useFifo48H (*C var*), 386
drivers_RT_RT600_MIMXRT685S._i2s_handle.userData (*C var*), 385
drivers_RT_RT600_MIMXRT685S._i2s_handle.watermark (*C var*), 386
drivers_RT_RT600_MIMXRT685S._i2s_master_slave (*C enum*), 382

drivers_RT_RT600_MIMXRT685S._i2s_master_slave.kI2S_MasterSlaveExtSckMaster (*C enumerator*), 382
drivers_RT_RT600_MIMXRT685S._i2s_master_slave.kI2S_MasterSlaveNormalMaster (*C enumerator*), 382
drivers_RT_RT600_MIMXRT685S._i2s_master_slave.kI2S_MasterSlaveNormalSlave (*C enumerator*), 382
drivers_RT_RT600_MIMXRT685S._i2s_master_slave.kI2S_MasterSlaveWsSyncMaster (*C enumerator*), 382
drivers_RT_RT600_MIMXRT685S._i2s_mode (*C enum*), 382
drivers_RT_RT600_MIMXRT685S._i2s_mode.kI2S_ModeDspWs50 (*C enumerator*), 383
drivers_RT_RT600_MIMXRT685S._i2s_mode.kI2S_ModeDspWsLong (*C enumerator*), 383
drivers_RT_RT600_MIMXRT685S._i2s_mode.kI2S_ModeDspWsShort (*C enumerator*), 383
drivers_RT_RT600_MIMXRT685S._i2s_mode.kI2S_ModeI2sClassic (*C enumerator*), 382
drivers_RT_RT600_MIMXRT685S._i2s_transfer (*C struct*), 385
drivers_RT_RT600_MIMXRT685S._i2s_transfer.data (*C var*), 385
drivers_RT_RT600_MIMXRT685S._i2s_transfer.dataSize (*C var*), 385
drivers_RT_RT600_MIMXRT685S._i3c_baudrate (*C struct*), 422
drivers_RT_RT600_MIMXRT685S._i3c_baudrate.i2cBaud (*C var*), 423
drivers_RT_RT600_MIMXRT685S._i3c_baudrate.i3cOpenDrainBaud (*C var*), 423
drivers_RT_RT600_MIMXRT685S._i3c_baudrate.i3cPushPullBaud (*C var*), 423
drivers_RT_RT600_MIMXRT685S._i3c_bus_request (*C enum*), 416
drivers_RT_RT600_MIMXRT685S._i3c_bus_request.kI3C_RequestAutoIbi (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_bus_request.kI3C_RequestEmitStartAddr (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_bus_request.kI3C_RequestEmitStop (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_bus_request.kI3C_RequestForceExit (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_bus_request.kI3C_RequestIbiAckNack (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_bus_request.kI3C_RequestNone (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_bus_request.kI3C_RequestProcessDAA (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_bus_type (*C enum*), 416
drivers_RT_RT600_MIMXRT685S._i3c_bus_type.kI3C_TypeI2C (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_bus_type.kI3C_TypeI3CDdr (*C enumerator*), 417
drivers_RT_RT600_MIMXRT685S._i3c_bus_type.kI3C_TypeI3CSdr (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_config (*C struct*), 390
drivers_RT_RT600_MIMXRT685S._i3c_config.baudRate_Hz (*C var*), 390
drivers_RT_RT600_MIMXRT685S._i3c_config.bcr (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.dcr (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.disableTimeout (*C var*), 390
drivers_RT_RT600_MIMXRT685S._i3c_config.enableMaster (*C var*), 390
drivers_RT_RT600_MIMXRT685S._i3c_config.enableOpenDrainHigh (*C var*), 390
drivers_RT_RT600_MIMXRT685S._i3c_config.enableOpenDrainStop (*C var*), 390
drivers_RT_RT600_MIMXRT685S._i3c_config.enableSlave (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.hdrMode (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.hKeep (*C var*), 390
drivers_RT_RT600_MIMXRT685S._i3c_config.ignoreS0S1Error (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.masterDynamicAddress (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.matchSlaveStartStop (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.maxReadLength (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.maxWriteLength (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.nakAllRequest (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.offline (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.partNumber (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.restartSclDelay (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.startSclDelay (*C var*), 390
drivers_RT_RT600_MIMXRT685S._i3c_config.staticAddr (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_config.vendorID (*C var*), 391
drivers_RT_RT600_MIMXRT685S._i3c_device_info (*C struct*), 388

drivers_RT_RT600_MIMXRT685S._i3c_device_info.bcr (*C var*), 388
drivers_RT_RT600_MIMXRT685S._i3c_device_info.dcr (*C var*), 388
drivers_RT_RT600_MIMXRT685S._i3c_device_info.dynamicAddr (*C var*), 388
drivers_RT_RT600_MIMXRT685S._i3c_device_info.hdrMode (*C var*), 389
drivers_RT_RT600_MIMXRT685S._i3c_device_info.maxReadLength (*C var*), 389
drivers_RT_RT600_MIMXRT685S._i3c_device_info.maxWriteLength (*C var*), 389
drivers_RT_RT600_MIMXRT685S._i3c_device_info.partNumber (*C var*), 388
drivers_RT_RT600_MIMXRT685S._i3c_device_info.staticAddr (*C var*), 388
drivers_RT_RT600_MIMXRT685S._i3c_device_info.vendorID (*C var*), 388
drivers_RT_RT600_MIMXRT685S._i3c_direction (*C enum*), 417
drivers_RT_RT600_MIMXRT685S._i3c_direction.kI3C_Read (*C enumerator*), 418
drivers_RT_RT600_MIMXRT685S._i3c_direction.kI3C_Write (*C enumerator*), 418
drivers_RT_RT600_MIMXRT685S._i3c_hdr_mode (*C enum*), 387
drivers_RT_RT600_MIMXRT685S._i3c_hdr_mode.kI3C_HDRModeDDR (*C enumerator*), 388
drivers_RT_RT600_MIMXRT685S._i3c_hdr_mode.kI3C_HDRModeNone (*C enumerator*), 388
drivers_RT_RT600_MIMXRT685S._i3c_hdr_mode.kI3C_HDRModeTSL (*C enumerator*), 388
drivers_RT_RT600_MIMXRT685S._i3c_hdr_mode.kI3C_HDRModeTSP (*C enumerator*), 388
drivers_RT_RT600_MIMXRT685S._i3c_ibi_response (*C enum*), 417
drivers_RT_RT600_MIMXRT685S._i3c_ibi_response.kI3C_IbiRespAck (*C enumerator*), 417
drivers_RT_RT600_MIMXRT685S._i3c_ibi_response.kI3C_IbiRespAckMandatory (*C enumerator*), 417
drivers_RT_RT600_MIMXRT685S._i3c_ibi_response.kI3C_IbiRespManual (*C enumerator*), 417
drivers_RT_RT600_MIMXRT685S._i3c_ibi_response.kI3C_IbiRespNack (*C enumerator*), 417
drivers_RT_RT600_MIMXRT685S._i3c_ibi_state (*C enum*), 417
drivers_RT_RT600_MIMXRT685S._i3c_ibi_state.kI3C_IbiAckNackPending (*C enumerator*), 417
drivers_RT_RT600_MIMXRT685S._i3c_ibi_state.kI3C_IbiDataBuffNeed (*C enumerator*), 417
drivers_RT_RT600_MIMXRT685S._i3c_ibi_state.kI3C_IbiReady (*C enumerator*), 417
drivers_RT_RT600_MIMXRT685S._i3c_ibi_type (*C enum*), 417
drivers_RT_RT600_MIMXRT685S._i3c_ibi_type.kI3C_IbiHotJoin (*C enumerator*), 417
drivers_RT_RT600_MIMXRT685S._i3c_ibi_type.kI3C_IbiMasterRequest (*C enumerator*), 417
drivers_RT_RT600_MIMXRT685S._i3c_ibi_type.kI3C_IbiNormal (*C enumerator*), 417
drivers_RT_RT600_MIMXRT685S._i3c_legacy_i2c_speed_mode (*C enum*), 419
drivers_RT_RT600_MIMXRT685S._i3c_legacy_i2c_speed_mode.kI3C_LegacyI2CFastMode (*C enumerator*), 419
drivers_RT_RT600_MIMXRT685S._i3c_legacy_i2c_speed_mode.kI3C_LegacyI2CFastModePlus (*C enumerator*), 419
drivers_RT_RT600_MIMXRT685S._i3c_legacy_i2c_speed_mode.kI3C_LegacyI2CStandardMode (*C enumerator*), 419
drivers_RT_RT600_MIMXRT685S._i3c_master_baudrate_config (*C struct*), 423
drivers_RT_RT600_MIMXRT685S._i3c_master_baudrate_config.busMode (*C var*), 423
drivers_RT_RT600_MIMXRT685S._i3c_master_baudrate_config.enableODHPP (*C var*), 423
drivers_RT_RT600_MIMXRT685S._i3c_master_baudrate_config.i2cBaud (*C var*), 423
drivers_RT_RT600_MIMXRT685S._i3c_master_baudrate_config.i3cOpenDrainBaud (*C var*), 423
drivers_RT_RT600_MIMXRT685S._i3c_master_baudrate_config.i3cPushPullBaud (*C var*), 423
drivers_RT_RT600_MIMXRT685S._i3c_master_bus_mode (*C enum*), 419
drivers_RT_RT600_MIMXRT685S._i3c_master_bus_mode.kI3C_LimitedBus (*C enumerator*), 419
drivers_RT_RT600_MIMXRT685S._i3c_master_bus_mode.kI3C_MixedFastBus (*C enumerator*), 419
drivers_RT_RT600_MIMXRT685S._i3c_master_bus_mode.kI3C_PureBus (*C enumerator*), 419
drivers_RT_RT600_MIMXRT685S._i3c_master_config (*C struct*), 424
drivers_RT_RT600_MIMXRT685S._i3c_master_config.baudRate_Hz (*C var*), 425
drivers_RT_RT600_MIMXRT685S._i3c_master_config.disableTimeout (*C var*), 424
drivers_RT_RT600_MIMXRT685S._i3c_master_config.enableMaster (*C var*), 424
drivers_RT_RT600_MIMXRT685S._i3c_master_config.enableOpenDrainHigh (*C var*), 425
drivers_RT_RT600_MIMXRT685S._i3c_master_config.enableOpenDrainStop (*C var*), 425
drivers_RT_RT600_MIMXRT685S._i3c_master_config.hKeep (*C var*), 424
drivers_RT_RT600_MIMXRT685S._i3c_master_config.restartSclDelay (*C var*), 425
drivers_RT_RT600_MIMXRT685S._i3c_master_config.startSclDelay (*C var*), 425

drivers_RT_RT600_MIMXRT685S._i3c_master_daa_baudrate (*C struct*), 423
drivers_RT_RT600_MIMXRT685S._i3c_master_daa_baudrate.i3cOpenDrainBaud (*C var*), 423
drivers_RT_RT600_MIMXRT685S._i3c_master_daa_baudrate.i3cPushPullBaud (*C var*), 423
drivers_RT_RT600_MIMXRT685S._i3c_master_daa_baudrate.sourceClock_Hz (*C var*), 423
drivers_RT_RT600_MIMXRT685S._i3c_master_dma_callback (*C struct*), 395
drivers_RT_RT600_MIMXRT685S._i3c_master_dma_handle (*C struct*), 395
drivers_RT_RT600_MIMXRT685S._i3c_master_enable (*C enum*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_enable.kI3C_MasterCapable (*C enumerator*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_enable.kI3C_MasterOff (*C enumerator*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_enable.kI3C_MasterOn (*C enumerator*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_error_flags (*C enum*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_error_flags.kI3C_MasterAllErrorFlags (*C enumerator*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_error_flags.kI3C_MasterErrorCrcFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_error_flags.kI3C_MasterErrorInvalidReqFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_error_flags.kI3C_MasterErrorMsgFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_error_flags.kI3C_MasterErrorNackFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_error_flags.kI3C_MasterErrorParityFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_error_flags.kI3C_MasterErrorReadFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_error_flags.kI3C_MasterErrorTimeoutFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_error_flags.kI3C_MasterErrorWriteAbortFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_error_flags.kI3C_MasterErrorWriteFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_flags (*C enum*), 413
drivers_RT_RT600_MIMXRT685S._i3c_master_flags.kI3C_MasterArbitrationWonFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_flags.kI3C_MasterBetweenFlag (*C enumerator*), 413
drivers_RT_RT600_MIMXRT685S._i3c_master_flags.kI3C_MasterClearFlags (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_flags.kI3C_MasterCompleteFlag (*C enumerator*), 413
drivers_RT_RT600_MIMXRT685S._i3c_master_flags.kI3C_MasterControlDoneFlag (*C enumerator*), 413
drivers_RT_RT600_MIMXRT685S._i3c_master_flags.kI3C_MasterErrorFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_flags.kI3C_MasterNackDetectFlag (*C enumerator*), 413
drivers_RT_RT600_MIMXRT685S._i3c_master_flags.kI3C_MasterRxReadyFlag (*C enumerator*), 413
drivers_RT_RT600_MIMXRT685S._i3c_master_flags.kI3C_MasterSlave2MasterFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_flags.kI3C_MasterSlaveStartFlag (*C enumerator*), 413
drivers_RT_RT600_MIMXRT685S._i3c_master_flags.kI3C_MasterTxReadyFlag (*C enumerator*), 414
drivers_RT_RT600_MIMXRT685S._i3c_master_handle (*C struct*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_handle.callback (*C var*), 427
drivers_RT_RT600_MIMXRT685S._i3c_master_handle.ibiAddress (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_handle.ibiBuff (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_handle.ibiPayloadSize (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_handle.ibiType (*C var*), 427

drivers_RT_RT600_MIMXRT685S._i3c_master_handle.remainingBytes (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_handle.rxTermOps (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_handle.state (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_handle.transfer (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_handle.userData (*C var*), 427
drivers_RT_RT600_MIMXRT685S._i3c_master_hkeep (*C enum*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_hkeep.kI3C_MasterHighKeeperNone (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_master_hkeep.kI3C_MasterHighKeeperWiredIn (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_master_hkeep.kI3C_MasterPassiveSDA (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_master_hkeep.kI3C_MasterPassiveSDASCL (*C enumerator*), 416
drivers_RT_RT600_MIMXRT685S._i3c_master_src_clk (*C enum*), 419
drivers_RT_RT600_MIMXRT685S._i3c_master_src_clk.kI3C_MasterSrcClk24M (*C enumerator*), 420
drivers_RT_RT600_MIMXRT685S._i3c_master_src_clk.kI3C_MasterSrcClk25M (*C enumerator*), 420
drivers_RT_RT600_MIMXRT685S._i3c_master_state (*C enum*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_state.kI3C_MasterStateDaa (*C enumerator*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_state.kI3C_MasterStateDdr (*C enumerator*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_state.kI3C_MasterStateIbiAck (*C enumerator*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_state.kI3C_MasterStateIbiRcv (*C enumerator*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_state.kI3C_MasterStateIdle (*C enumerator*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_state.kI3C_MasterStateMsgSdr (*C enumerator*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_state.kI3C_MasterStateNormAct (*C enumerator*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_state.kI3C_MasterStateSlvReq (*C enumerator*), 415
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer (*C struct*), 425
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_callback (*C struct*), 425
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_callback.ibiCallback (*C var*), 425
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_callback.slave2Master (*C var*), 425
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_callback.transferComplete (*C var*), 425
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_flags (*C enum*), 420
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_flags.kI3C_TransferDefaultFlag (*C enumerator*), 420
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_flags.kI3C_TransferDisableRxTermFlag (*C enumerator*), 420
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_flags.kI3C_TransferNoStartFlag (*C enumerator*), 420
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_flags.kI3C_TransferNoStopFlag (*C enumerator*), 420
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_flags.kI3C_TransferRepeatedStartFlag (*C enumerator*), 420
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_flags.kI3C_TransferRxAutoTermFlag (*C enumerator*), 420
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_flags.kI3C_TransferStartWithBroadcastAddr (*C enumerator*), 420
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer_flags.kI3C_TransferWordsFlag (*C enumerator*), 420
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer.busType (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer.data (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer.dataSize (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer.direction (*C var*), 425

drivers_RT_RT600_MIMXRT685S._i3c_master_transfer.flags (*C var*), 425
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer.ibiResponse (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer.slaveAddress (*C var*), 425
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer.subaddress (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_master_transfer.subaddressSize (*C var*), 426
drivers_RT_RT600_MIMXRT685S._i3c_register_ibi_addr (*C struct*), 422
drivers_RT_RT600_MIMXRT685S._i3c_register_ibi_addr.address (*C var*), 422
drivers_RT_RT600_MIMXRT685S._i3c_register_ibi_addr.i3cFastStart (*C var*), 422
drivers_RT_RT600_MIMXRT685S._i3c_register_ibi_addr.ibiHasPayload (*C var*), 422
drivers_RT_RT600_MIMXRT685S._i3c_rx_term_ops (*C enum*), 418
drivers_RT_RT600_MIMXRT685S._i3c_rx_term_ops.kI3C_RxAutoTerm (*C enumerator*), 418
drivers_RT_RT600_MIMXRT685S._i3c_rx_term_ops.kI3C_RxTermDisable (*C enumerator*), 418
drivers_RT_RT600_MIMXRT685S._i3c_rx_term_ops.kI3C_RxTermLastByte (*C enumerator*), 419
drivers_RT_RT600_MIMXRT685S._i3c_rx_trigger_level (*C enum*), 418
drivers_RT_RT600_MIMXRT685S._i3c_rx_trigger_level.kI3C_RxTriggerOnNotEmpty (*C enumerator*), 418
drivers_RT_RT600_MIMXRT685S._i3c_rx_trigger_level.kI3C_RxTriggerUntilOneHalfOrMore (*C enumerator*), 418
drivers_RT_RT600_MIMXRT685S._i3c_rx_trigger_level.kI3C_RxTriggerUntilOneQuarterOrMore (*C enumerator*), 418
drivers_RT_RT600_MIMXRT685S._i3c_rx_trigger_level.kI3C_RxTriggerUntilThreeQuarterOrMore (*C enumerator*), 418
drivers_RT_RT600_MIMXRT685S._i3c_slave_activity_state (*C enum*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_activity_state.kI3C_SlaveLatency1Ms (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_activity_state.kI3C_SlaveLatency10S (*C enumerator*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_activity_state.kI3C_SlaveLatency100Ms (*C enumerator*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_activity_state.kI3C_SlaveNoLatency (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_config (*C struct*), 440
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.bcr (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.dcr (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.enableSlave (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.hdrMode (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.ignoreS0S1Error (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.matchSlaveStartStop (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.maxReadLength (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.maxWriteLength (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.nakAllRequest (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.offline (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.partNumber (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.staticAddr (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_config.vendorID (*C var*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_dma_handle (*C struct*), 428
drivers_RT_RT600_MIMXRT685S._i3c_slave_dma_transfer (*C struct*), 428
drivers_RT_RT600_MIMXRT685S._i3c_slave_error_flags (*C enum*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_error_flags.kI3C_SlaveErrorHdrCRCFlag (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_error_flags.kI3C_SlaveErrorHdrParityFlag (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_error_flags.kI3C_SlaveErrorInvalidStartFlag (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_error_flags.kI3C_SlaveErrorOverreadFlag (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_error_flags.kI3C_SlaveErrorOverrunFlag (*C enumerator*), 437

drivers_RT_RT600_MIMXRT685S._i3c_slave_error_flags.kI3C_SlaveErrorOverwriteFlag (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_error_flags.kI3C_SlaveErrorS0S1Flag (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_error_flags.kI3C_SlaveErrorSdrParityFlag (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_error_flags.kI3C_SlaveErrorTermFlag (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_error_flags.kI3C_SlaveErrorUnderrunFlag (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_error_flags.kI3C_SlaveErrorUnderrunNakFlag (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_event (*C enum*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_event.kI3C_SlaveEventHotJoinReq (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_event.kI3C_SlaveEventIBI (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_event.kI3C_SlaveEventMasterReq (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_event.kI3C_SlaveEventNormal (*C enumerator*), 438
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags (*C enum*), 435
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveAllIrqFlags (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveBusDAAFlag (*C enumerator*), 436
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveBusHDRModeFlag (*C enumerator*), 436
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveBusStartFlag (*C enumerator*), 436
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveBusStopFlag (*C enumerator*), 436
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveCCCHandledFlag (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveClearFlags (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveDynamicAddrChangedFlag (*C enumerator*), 436
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveErrorFlag (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveEventSentFlag (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveHDRCommandMatchFlag (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveHotJoinDisabledFlag (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveIbiDisableFlag (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveMasterRequestDisabledFlag (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveMatchedFlag (*C enumerator*), 436
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveMessageFlag (*C enumerator*), 436
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveNotStopFlag (*C enumerator*), 436
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveReceivedCCCFlag (*C enumerator*), 437
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveRequiredReadFlag (*C enumerator*), 436
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveRequiredWriteFlag (*C enumerator*), 436
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveRxReadyFlag (*C enumerator*), 436
drivers_RT_RT600_MIMXRT685S._i3c_slave_flags.kI3C_SlaveTxReadyFlag (*C enumerator*), 436
drivers_RT_RT600_MIMXRT685S._i3c_slave_handle (*C struct*), 442
drivers_RT_RT600_MIMXRT685S._i3c_slave_handle.callback (*C var*), 443
drivers_RT_RT600_MIMXRT685S._i3c_slave_handle.eventMask (*C var*), 442
drivers_RT_RT600_MIMXRT685S._i3c_slave_handle.isBusy (*C var*), 442
drivers_RT_RT600_MIMXRT685S._i3c_slave_handle.transfer (*C var*), 442
drivers_RT_RT600_MIMXRT685S._i3c_slave_handle.transferredCount (*C var*), 442
drivers_RT_RT600_MIMXRT685S._i3c_slave_handle.txFifoSize (*C var*), 443

drivers_RT_RT600_MIMXRT685S._i3c_slave_handle.userData (*C var*), 443
drivers_RT_RT600_MIMXRT685S._i3c_slave_handle.wasTransmit (*C var*), 442
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer (*C struct*), 441
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer_event (*C enum*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer_event.kI3C_SlaveAddressMatchEvent (*C enumerator*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer_event.kI3C_SlaveAllEvents (*C enumerator*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer_event.kI3C_SlaveCompletionEvent (*C enumerator*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer_event.kI3C_SlaveHDRCommandMatchEvent (*C enumerator*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer_event.kI3C_SlaveReceivedCCCEvent (*C enumerator*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer_event.kI3C_SlaveReceiveEvent (*C enumerator*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer_event.kI3C_SlaveRequestSentEvent (*C enumerator*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer_event.kI3C_SlaveRequiredTransmitEvent (*C enumerator*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer_event.kI3C_SlaveStartEvent (*C enumerator*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer_event.kI3C_SlaveTransmitEvent (*C enumerator*), 439
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer.completionStatus (*C var*), 442
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer.event (*C var*), 442
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer.rxData (*C var*), 442
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer.rxDataSize (*C var*), 442
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer.transferredCount (*C var*), 442
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer.txData (*C var*), 442
drivers_RT_RT600_MIMXRT685S._i3c_slave_transfer.txDataSize (*C var*), 442
drivers_RT_RT600_MIMXRT685S._i3c_start_scl_delay (*C enum*), 419
drivers_RT_RT600_MIMXRT685S._i3c_start_scl_delay.kI3C_IncreaseSclHalfPeriod (*C enumerator*), 419
drivers_RT_RT600_MIMXRT685S._i3c_start_scl_delay.kI3C_IncreaseSclOneAndHalfPeriod (*C enumerator*), 419
drivers_RT_RT600_MIMXRT685S._i3c_start_scl_delay.kI3C_IncreaseSclOnePeriod (*C enumerator*), 419
drivers_RT_RT600_MIMXRT685S._i3c_start_scl_delay.kI3C_NoDelay (*C enumerator*), 419
drivers_RT_RT600_MIMXRT685S._i3c_timing (*C struct*), 423
drivers_RT_RT600_MIMXRT685S._i3c_timing.actualI2cBaud (*C var*), 424
drivers_RT_RT600_MIMXRT685S._i3c_timing.actualOdBaud (*C var*), 424
drivers_RT_RT600_MIMXRT685S._i3c_timing.actualPpBaud (*C var*), 424
drivers_RT_RT600_MIMXRT685S._i3c_timing.i2cBaud (*C var*), 424
drivers_RT_RT600_MIMXRT685S._i3c_timing.odBaud (*C var*), 424
drivers_RT_RT600_MIMXRT685S._i3c_timing.odHighPP (*C var*), 424
drivers_RT_RT600_MIMXRT685S._i3c_timing.ppBaud (*C var*), 424
drivers_RT_RT600_MIMXRT685S._i3c_timing.ppLow (*C var*), 424
drivers_RT_RT600_MIMXRT685S._i3c_tx_trigger_level (*C enum*), 418
drivers_RT_RT600_MIMXRT685S._i3c_tx_trigger_level.kI3C_TxTriggerOnEmpty (*C enumerator*), 418
drivers_RT_RT600_MIMXRT685S._i3c_tx_trigger_level.kI3C_TxTriggerUntilOneHalfOrLess (*C enumerator*), 418
drivers_RT_RT600_MIMXRT685S._i3c_tx_trigger_level.kI3C_TxTriggerUntilOneLessThanFull (*C enumerator*), 418
drivers_RT_RT600_MIMXRT685S._i3c_tx_trigger_level.kI3C_TxTriggerUntilOneQuarterOrLess (*C enumerator*), 418
drivers_RT_RT600_MIMXRT685S._iap_boot_option (*C struct*), 443

drivers_RT_RT600_MIMXRT685S._iap_boot_option.option (*C union*), 443
drivers_RT_RT600_MIMXRT685S._iap_boot_option.option.B (*C struct*), 443
drivers_RT_RT600_MIMXRT685S._iap_boot_option.option.B (*C var*), 443
drivers_RT_RT600_MIMXRT685S._iap_boot_option.option.B.bootImageIndex (*C var*), 444
drivers_RT_RT600_MIMXRT685S._iap_boot_option.option.B.bootInterface (*C var*), 444
drivers_RT_RT600_MIMXRT685S._iap_boot_option.option.B.instance (*C var*), 444
drivers_RT_RT600_MIMXRT685S._iap_boot_option.option.B.mode (*C var*), 444
drivers_RT_RT600_MIMXRT685S._iap_boot_option.option.U (*C var*), 443
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t (*C enum*), 462
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_AcmpToDma0 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_AcmpToDma1 (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_AdcIrqToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_AdcToDma0 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_AdcToDma1 (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_ArmTxevToSct0 (*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Cmp0OutToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer0M0ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer0M0ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer0M1ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer0M1ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer0Mat0ToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer0ToDspInterrupt (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer1M0ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer1M0ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer1M1ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer1M1ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer1Mat0ToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer1ToDspInterrupt (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer2M0ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer2M0ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer2M1ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer2M1ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer2Mat0ToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer2ToDspInterrupt

(*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer3M0ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer3M0ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer3M1ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer3M1ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer3Mat0ToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer3ToDspInterrupt (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer4M0ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer4M0ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer4M1ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer4M1ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer4Mat0ToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Ctimer4ToDspInterrupt (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp0ToTimer0CaptureChannels (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp0ToTimer1CaptureChannels (*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp0ToTimer2CaptureChannels (*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp0ToTimer3CaptureChannels (*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp0ToTimer4CaptureChannels (*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp1ToTimer0CaptureChannels (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp1ToTimer1CaptureChannels (*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp1ToTimer2CaptureChannels (*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp1ToTimer3CaptureChannels (*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp1ToTimer4CaptureChannels (*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp2ToTimer0CaptureChannels (*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp2ToTimer1CaptureChannels (*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp2ToTimer2CaptureChannels (*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp2ToTimer3CaptureChannels (*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp2ToTimer4CaptureChannels (*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp3ToTimer0CaptureChannels (*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp3ToTimer1CaptureChannels

(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp3ToTimer2CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp3ToTimer3CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp3ToTimer4CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp4ToTimer0CaptureChannels
(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp4ToTimer1CaptureChannels
(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp4ToTimer2CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp4ToTimer3CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp4ToTimer4CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp5ToTimer0CaptureChannels
(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp5ToTimer1CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp5ToTimer2CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp5ToTimer3CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp5ToTimer4CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp6ToTimer0CaptureChannels
(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp6ToTimer1CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp6ToTimer2CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp6ToTimer3CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp6ToTimer4CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp7ToTimer0CaptureChannels
(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp7ToTimer1CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp7ToTimer2CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp7ToTimer3CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp7ToTimer4CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp8ToTimer0CaptureChannels
(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp8ToTimer1CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp8ToTimer2CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp8ToTimer3CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp8ToTimer4CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp9ToTimer0CaptureChannels

(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp9ToTimer1CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp9ToTimer2CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp9ToTimer3CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp9ToTimer4CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp10ToTimer0CaptureChannels
(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp10ToTimer1CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp10ToTimer2CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp10ToTimer3CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp10ToTimer4CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp11ToTimer0CaptureChannels
(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp11ToTimer1CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp11ToTimer2CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp11ToTimer3CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp11ToTimer4CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp12ToTimer0CaptureChannels
(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp12ToTimer1CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp12ToTimer2CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp12ToTimer3CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp12ToTimer4CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp13ToTimer0CaptureChannels
(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp13ToTimer1CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp13ToTimer2CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp13ToTimer3CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp13ToTimer4CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp14ToTimer0CaptureChannels
(*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp14ToTimer1CaptureChannels
(*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp14ToTimer2CaptureChannels
(*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp14ToTimer3CaptureChannels
(*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp14ToTimer4CaptureChannels

(*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp15ToTimer0CaptureChannels (*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp15ToTimer1CaptureChannels (*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp15ToTimer2CaptureChannels (*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp15ToTimer3CaptureChannels (*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_CtInp15ToTimer4CaptureChannels (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_DebugHaltedToSct0 (*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel0ToTriginChannels (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel1ToTriginChannels (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel2ToTriginChannels (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel3ToTriginChannels (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel4ToTriginChannels (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel5ToTriginChannels (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel6ToTriginChannels (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel7ToTriginChannels (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel8ToTriginChannels (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel9ToTriginChannels (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel10ToTriginChannels (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel11ToTriginChannels (*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel12ToTriginChannels (*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel13ToTriginChannels (*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel14ToTriginChannels (*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel15ToTriginChannels (*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel16ToTriginChannels (*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel17ToTriginChannels (*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel18ToTriginChannels (*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel19ToTriginChannels (*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel20ToTriginChannels (*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel21ToTriginChannels (*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel22ToTriginChannels

(*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel23ToTriginChannels
(*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel24ToTriginChannels
(*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel25ToTriginChannels
(*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel26ToTriginChannels
(*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel27ToTriginChannels
(*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel28ToTriginChannels
(*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel29ToTriginChannels
(*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel30ToTriginChannels
(*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel31ToTriginChannels
(*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0OtrigChannel32ToTriginChannels
(*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0TrigOutAToDma0
(*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0TrigOutBToDma0
(*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0TrigOutCToDma0
(*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma0TrigOutDToDma0
(*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel0ToTriginChannels
(*C enumerator*), 476
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel1ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel2ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel3ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel4ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel5ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel6ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel7ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel8ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel9ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel10ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel11ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel12ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel13ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel14ToTriginChannels

(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel15ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel16ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel17ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel18ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel19ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel20ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel21ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel22ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel23ToTriginChannels
(*C enumerator*), 477
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel24ToTriginChannels
(*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel25ToTriginChannels
(*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel26ToTriginChannels
(*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel27ToTriginChannels
(*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel28ToTriginChannels
(*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel29ToTriginChannels
(*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel30ToTriginChannels
(*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel31ToTriginChannels
(*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1OtrigChannel32ToTriginChannels
(*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1TrigOutAToDma1
(*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1TrigOutBToDma1
(*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1TrigOutCToDma1
(*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dma1TrigOutDToDma1
(*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dmac0ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dmac1ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Dmic0ToDspInterrupt
(*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_FfroToFreqmeas (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Flexcomm0ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Flexcomm1ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Flexcomm2ToDspInterrupt

(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Flexcomm3ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Flexcomm4ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Flexcomm5ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Flexcomm6ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Flexcomm7ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_FlexspiRxToDma0 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_FlexspiRxToDma1 (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_FlexspiToDspInterrupt
(*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_FlexspiTxToDma0 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_FlexspiTxToDma1 (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_FreqmeGpioClkToFreqmeas
(*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioInt0ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioInt1ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioInt2ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioInt3ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioInt4ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioInt5ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioInt6ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioInt7ToDspInterrupt
(*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioIntBmatchToSct0
(*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin0ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin1ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin2ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin3ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin4ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin5ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin6ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin7ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin8ToPintsel

(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin9ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin10ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin11ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin12ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin13ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin14ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin15ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin16ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin17ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin18ToPintsel
(*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin19ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin20ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin21ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin22ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin23ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin24ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin25ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin26ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin27ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin28ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin29ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin30ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort0Pin31ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin0ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin1ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin2ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin3ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin4ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin5ToPintsel

(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin6ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin7ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin8ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin9ToPintsel
(*C enumerator*), 465
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin10ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin11ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin12ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin13ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin14ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin15ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin16ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin17ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin18ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin19ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin20ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin21ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin22ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin23ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin24ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin25ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin26ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin27ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin28ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin29ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin30ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_GpioPort1Pin31ToPintsel
(*C enumerator*), 466
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_HashCryptOutToDma0
(*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_HashCryptOutToDma1
(*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Hwvad0ToDspInterrupt

(*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_I3c0ToDspInterrupt (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_LposcToFreqmeas (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_MainSysClkToFreqmeas (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_MclkToSct0 (*C enumerator*), 464
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Mrt0ToDspInterrupt (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_MuBToDspInterrupt (*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_NsGpioPint0Int0ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_NsGpioPint0Int0ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_NsGpioPint0Int1ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_NsGpioPint0Int1ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_NsGpioPint0Int2ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_NsGpioPint0Int2ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_NsGpioPint0Int3ToDma0 (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_NsGpioPint0Int3ToDma1 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_NsHsGpioInt0ToDspInterrupt (*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_NsHsGpioInt1ToDspInterrupt (*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_OsEventTimerToDspInterrupt (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Rtc32KhzOscToFreqmeas (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_RtcToDspInterrupt (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Sct0Dmac0ToDma0 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Sct0Dmac0ToDma1 (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Sct0Dmac1ToDma0 (*C enumerator*), 474
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Sct0Dmac1ToDma1 (*C enumerator*), 475
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Sct0PinInp0ToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Sct0PinInp1ToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Sct0PinInp2ToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Sct0PinInp3ToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Sct0PinInp4ToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Sct0PinInp5ToSct0 (*C*

enumerator), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Sct0PinInp6ToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Sct0PinInp7ToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SfroToFreqmeas (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s0SclkToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s0WsToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s0WsToTimer0CaptureChannels (*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s0WsToTimer1CaptureChannels (*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s0WsToTimer2CaptureChannels (*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s0WsToTimer3CaptureChannels (*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s0WsToTimer4CaptureChannels (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s1SclkToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s1WsToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s1WsToTimer0CaptureChannels (*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s1WsToTimer1CaptureChannels (*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s1WsToTimer2CaptureChannels (*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s1WsToTimer3CaptureChannels (*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_SharedI2s1WsToTimer4CaptureChannels (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Usb1FrameToggleToSct0 (*C enumerator*), 463
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Usb1FrameToggleToTimer0CaptureChan (*C enumerator*), 469
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Usb1FrameToggleToTimer1CaptureChan (*C enumerator*), 470
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Usb1FrameToggleToTimer2CaptureChan (*C enumerator*), 471
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Usb1FrameToggleToTimer3CaptureChan (*C enumerator*), 472
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Usb1FrameToggleToTimer4CaptureChan (*C enumerator*), 473
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Utick0ToDspInterrupt (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_Wdt1ToDspInterrupt (*C enumerator*), 467
drivers_RT_RT600_MIMXRT685S._inputmux_connection_t.kINPUTMUX_XtalinToFreqmeas (*C enumerator*), 468
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t (*C enum*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerAcmpEna (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerAdcEna (*C enumerator*), 479

drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerCtimer0M0Ena (*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerCtimer0M1Ena (*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerCtimer1M0Ena (*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerCtimer1M1Ena (*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerCtimer2M0Ena (*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerCtimer2M1Ena (*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerCtimer3M0Ena (*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerCtimer3M1Ena (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerCtimer4M0Ena (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerCtimer4M1Ena (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerDma0OutAEna (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerDma0OutBEna (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerDma0OutCEna (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerDma0OutDEna (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerFlexspiRxEna (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerFlexspiTxEna (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerHashOutEna (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerPint0Ena (*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerPint1Ena (*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerPint2Ena (*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerPint3Ena (*C enumerator*), 478
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerSctDmac0Ena (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac0InputTriggerSctDmac1Ena (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerAcmpEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerAdcEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerCtimer0M0Ena (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerCtimer0M1Ena (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerCtimer1M0Ena (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerCtimer1M1Ena (*C enumerator*), 479

drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerCtimer2M0Ena (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerCtimer2M1Ena (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerCtimer3M0Ena (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerCtimer3M1Ena (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerCtimer4M0Ena (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerCtimer4M1Ena (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerDma1OutAEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerDma1OutBEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerDma1OutCEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerDma1OutDEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerFlexspiRxEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerFlexspiTxEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerHashOutEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerPint0Ena (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerPint1Ena (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerPint2Ena (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerPint3Ena (*C enumerator*), 479
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerSctDmac0Ena (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmac1InputTriggerSctDmac1Ena (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch0ToDmac0Ch16RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch0ToDmac1Ch16RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch1ToDmac0Ch17RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch1ToDmac1Ch17RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch2ToDmac0Ch18RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch2ToDmac1Ch18RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch3ToDmac0Ch19RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch3ToDmac1Ch19RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch4ToDmac0Ch20RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch4ToDmac1Ch20RequestEna (*C enumerator*), 482

drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch5ToDmac0Ch21RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch5ToDmac1Ch21RequestEna (*C enumerator*), 483
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch6ToDmac0Ch22RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch6ToDmac1Ch22RequestEna (*C enumerator*), 483
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch7ToDmac0Ch23RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Dmic0Ch7ToDmac1Ch23RequestEna (*C enumerator*), 483
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm0RxToDmac0Ch0RequestEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm0RxToDmac1Ch0RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm0TxToDmac0Ch1RequestEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm0TxToDmac1Ch1RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm1RxToDmac0Ch2RequestEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm1RxToDmac1Ch2RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm1TxToDmac0Ch3RequestEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm1TxToDmac1Ch3RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm2RxToDmac0Ch4RequestEna (*C enumerator*), 480
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm2RxToDmac1Ch4RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm2TxToDmac0Ch5RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm2TxToDmac1Ch5RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm3RxToDmac0Ch6RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm3RxToDmac1Ch6RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm3TxToDmac0Ch7RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm3TxToDmac1Ch7RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm4RxToDmac0Ch8RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm4RxToDmac1Ch8RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm4TxToDmac0Ch9RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm4TxToDmac1Ch9RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm5RxToDmac0Ch10RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm5RxToDmac1Ch10RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm5TxToDmac0Ch11RequestEna (*C enumerator*), 481

drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm5TxToDmac1Ch11RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm6RxToDmac0Ch12RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm6RxToDmac1Ch12RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm6TxToDmac0Ch13RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm6TxToDmac1Ch13RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm7RxToDmac0Ch14RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm7RxToDmac1Ch14RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm7TxToDmac0Ch15RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm7TxToDmac1Ch15RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm14RxToDmac0Ch26RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm14RxToDmac1Ch26RequestEna (*C enumerator*), 483
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm14TxToDmac0Ch27RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_Flexcomm14TxToDmac1Ch27RequestEna (*C enumerator*), 483
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_HashCryptToDmac0Ch30RequestEna (*C enumerator*), 482
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_HashCryptToDmac1Ch30RequestEna (*C enumerator*), 483
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_I3c0RxToDmac0Ch24RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_I3c0RxToDmac1Ch24RequestEna (*C enumerator*), 483
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_I3c0TxToDmac0Ch25RequestEna (*C enumerator*), 481
drivers_RT_RT600_MIMXRT685S._inputmux_signal_t.kINPUTMUX_I3c0TxToDmac1Ch25RequestEna (*C enumerator*), 483
drivers_RT_RT600_MIMXRT685S._iopctl_group (*C struct*), 488
drivers_RT_RT600_MIMXRT685S._lpadc_calibration_value (*C struct*), 527
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_average_mode (*C enum*), 510
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_average_mode.kLPADC_ConversionAverage1 (*C enumerator*), 510
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_average_mode.kLPADC_ConversionAverage2 (*C enumerator*), 510
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_average_mode.kLPADC_ConversionAverage4 (*C enumerator*), 510
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_average_mode.kLPADC_ConversionAverage8 (*C enumerator*), 510
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_average_mode.kLPADC_ConversionAverage16 (*C enumerator*), 510
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_average_mode.kLPADC_ConversionAverage32 (*C enumerator*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_average_mode.kLPADC_ConversionAverage64 (*C enumerator*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_average_mode.kLPADC_ConversionAverage128 (*C enumerator*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_average_mode.kLPADC_ConversionAverageMax

(*C enumerator*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_resolution_mode (*C enum*), 510
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_resolution_mode.kLPADC_ConversionResolutionHigh (*C enumerator*), 510
drivers_RT_RT600_MIMXRT685S._lpadc_conversion_resolution_mode.kLPADC_ConversionResolutionStandard (*C enumerator*), 510
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_average_mode (*C enum*), 508
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount1 (*C enumerator*), 508
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount2 (*C enumerator*), 508
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount4 (*C enumerator*), 508
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount8 (*C enumerator*), 508
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount16 (*C enumerator*), 508
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount32 (*C enumerator*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount64 (*C enumerator*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_average_mode.kLPADC_HardwareAverageCount128 (*C enumerator*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_compare_mode (*C enum*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_compare_mode.kLPADC_HardwareCompareDisabled (*C enumerator*), 510
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_compare_mode.kLPADC_HardwareCompareRepeatUntilTrue (*C enumerator*), 510
drivers_RT_RT600_MIMXRT685S._lpadc_hardware_compare_mode.kLPADC_HardwareCompareStoreOnTrue (*C enumerator*), 510
drivers_RT_RT600_MIMXRT685S._lpadc_interrupt_enable (*C enum*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_interrupt_enable.kLPADC_FIFO0WatermarkInterruptEnable (*C enumerator*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_interrupt_enable.kLPADC_FIFOWatermarkInterruptEnable (*C enumerator*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_interrupt_enable.kLPADC_ResultFIFO0OverflowInterruptEnable (*C enumerator*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_interrupt_enable.kLPADC_ResultFIFOOverflowInterruptEnable (*C enumerator*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_interrupt_enable.kLPADC_Trigger0CompletionInterruptEnable (*C enumerator*), 507
drivers_RT_RT600_MIMXRT685S._lpadc_interrupt_enable.kLPADC_Trigger1CompletionInterruptEnable (*C enumerator*), 507
drivers_RT_RT600_MIMXRT685S._lpadc_interrupt_enable.kLPADC_TriggerExceptionInterruptEnable (*C enumerator*), 507
drivers_RT_RT600_MIMXRT685S._lpadc_offset_calibration_mode (*C enum*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_offset_calibration_mode.kLPADC_OffsetCalibration12bitMode (*C enumerator*), 512
drivers_RT_RT600_MIMXRT685S._lpadc_offset_calibration_mode.kLPADC_OffsetCalibration16bitMode (*C enumerator*), 512
drivers_RT_RT600_MIMXRT685S._lpadc_power_level_mode (*C enum*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_power_level_mode.kLPADC_PowerLevelAlt1 (*C enumerator*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_power_level_mode.kLPADC_PowerLevelAlt2 (*C enumerator*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_power_level_mode.kLPADC_PowerLevelAlt3 (*C enumerator*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_power_level_mode.kLPADC_PowerLevelAlt4 (*C enu-*

merator), 511
drivers_RT_RT600_MIMXRT685S._lpadc_reference_voltage_mode (*C enum*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_reference_voltage_mode.kLPADC_ReferenceVoltageAlt1 (*C enumerator*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_reference_voltage_mode.kLPADC_ReferenceVoltageAlt2 (*C enumerator*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_reference_voltage_mode.kLPADC_ReferenceVoltageAlt3 (*C enumerator*), 511
drivers_RT_RT600_MIMXRT685S._lpadc_sample_channel_mode (*C enum*), 507
drivers_RT_RT600_MIMXRT685S._lpadc_sample_channel_mode.kLPADC_SampleChannelDiffBothSide (*C enumerator*), 508
drivers_RT_RT600_MIMXRT685S._lpadc_sample_channel_mode.kLPADC_SampleChannelDiffBothSideAB (*C enumerator*), 508
drivers_RT_RT600_MIMXRT685S._lpadc_sample_channel_mode.kLPADC_SampleChannelDiffBothSideBA (*C enumerator*), 508
drivers_RT_RT600_MIMXRT685S._lpadc_sample_channel_mode.kLPADC_SampleChannelDualSingleEndBothSide (*C enumerator*), 508
drivers_RT_RT600_MIMXRT685S._lpadc_sample_channel_mode.kLPADC_SampleChannelSingleEndSideA (*C enumerator*), 508
drivers_RT_RT600_MIMXRT685S._lpadc_sample_channel_mode.kLPADC_SampleChannelSingleEndSideB (*C enumerator*), 508
drivers_RT_RT600_MIMXRT685S._lpadc_sample_scale_mode (*C enum*), 507
drivers_RT_RT600_MIMXRT685S._lpadc_sample_scale_mode.kLPADC_SampleFullScale (*C enumerator*), 507
drivers_RT_RT600_MIMXRT685S._lpadc_sample_scale_mode.kLPADC_SamplePartScale (*C enumerator*), 507
drivers_RT_RT600_MIMXRT685S._lpadc_sample_time_mode (*C enum*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK3 (*C enumerator*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK5 (*C enumerator*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK7 (*C enumerator*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK11 (*C enumerator*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK19 (*C enumerator*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK35 (*C enumerator*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK67 (*C enumerator*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_sample_time_mode.kLPADC_SampleTimeADCK131 (*C enumerator*), 509
drivers_RT_RT600_MIMXRT685S._lpadc_status_flags (*C enum*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_status_flags.kLPADC_ActiveFlag (*C enumerator*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_status_flags.kLPADC_CalibrationReadyFlag (*C enumerator*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_status_flags.kLPADC_ResultFIFO0OverflowFlag (*C enumerator*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_status_flags.kLPADC_ResultFIFO0ReadyFlag (*C enumerator*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_status_flags.kLPADC_ResultFIFOOverflowFlag (*C enumerator*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_status_flags.kLPADC_ResultFIFOReadyFlag (*C enumerator*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_status_flags.kLPADC_TriggerCompletionFlag (*C enumerator*), 506
drivers_RT_RT600_MIMXRT685S._lpadc_status_flags.kLPADC_TriggerExceptionFlag (*C enumer-*

ator), 506
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_priority_policy (*C enum*), 512
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_priority_policy.kLPADC_ConvPreemptImmediatelyAutoRestarte (*C enumerator*), 512
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_priority_policy.kLPADC_ConvPreemptImmediatelyAutoResumed (*C enumerator*), 512
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_priority_policy.kLPADC_ConvPreemptImmediatelyNotAutoResu (*C enumerator*), 512
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_priority_policy.kLPADC_ConvPreemptSoftlyAutoRestarted (*C enumerator*), 512
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_priority_policy.kLPADC_ConvPreemptSoftlyAutoResumed (*C enumerator*), 512
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_priority_policy.kLPADC_ConvPreemptSoftlyNotAutoResumed (*C enumerator*), 512
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_priority_policy.kLPADC_TriggerPriorityExceptionDisabled (*C enumerator*), 513
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_priority_policy.kLPADC_TriggerPriorityPreemptImmediately (*C enumerator*), 513
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_priority_policy.kLPADC_TriggerPriorityPreemptSoftly (*C enumerator*), 513
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_status_flags (*C enum*), 507
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_status_flags.kLPADC_Trigger0CompletedFlag (*C enumerator*), 507
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_status_flags.kLPADC_Trigger0InterruptedFlag (*C enumerator*), 507
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_status_flags.kLPADC_Trigger1CompletedFlag (*C enumerator*), 507
drivers_RT_RT600_MIMXRT685S._lpadc_trigger_status_flags.kLPADC_Trigger1InterruptedFlag (*C enumerator*), 507
drivers_RT_RT600_MIMXRT685S._lpadc_tune_value (*C enum*), 513
drivers_RT_RT600_MIMXRT685S._lpadc_tune_value.kLPADC_TuneValue0 (*C enumerator*), 513
drivers_RT_RT600_MIMXRT685S._lpadc_tune_value.kLPADC_TuneValue1 (*C enumerator*), 513
drivers_RT_RT600_MIMXRT685S._lpadc_tune_value.kLPADC_TuneValue2 (*C enumerator*), 513
drivers_RT_RT600_MIMXRT685S._lpadc_tune_value.kLPADC_TuneValue3 (*C enumerator*), 513
drivers_RT_RT600_MIMXRT685S._lut_sequence (*C struct*), 454
drivers_RT_RT600_MIMXRT685S._lut_sequence.seqId (*C var*), 454
drivers_RT_RT600_MIMXRT685S._lut_sequence.seqNum (*C var*), 454
drivers_RT_RT600_MIMXRT685S._mrt_chnl (*C enum*), 534
drivers_RT_RT600_MIMXRT685S._mrt_chnl.kMRT_Channel_0 (*C enumerator*), 534
drivers_RT_RT600_MIMXRT685S._mrt_chnl.kMRT_Channel_1 (*C enumerator*), 534
drivers_RT_RT600_MIMXRT685S._mrt_chnl.kMRT_Channel_2 (*C enumerator*), 534
drivers_RT_RT600_MIMXRT685S._mrt_chnl.kMRT_Channel_3 (*C enumerator*), 534
drivers_RT_RT600_MIMXRT685S._mrt_config (*C struct*), 535
drivers_RT_RT600_MIMXRT685S._mrt_config.enableMultiTask (*C var*), 536
drivers_RT_RT600_MIMXRT685S._mrt_interrupt_enable (*C enum*), 535
drivers_RT_RT600_MIMXRT685S._mrt_interrupt_enable.kMRT_TimerInterruptEnable (*C enumerator*), 535
drivers_RT_RT600_MIMXRT685S._mrt_status_flags (*C enum*), 535
drivers_RT_RT600_MIMXRT685S._mrt_status_flags.kMRT_TimerInterruptFlag (*C enumerator*), 535
drivers_RT_RT600_MIMXRT685S._mrt_status_flags.kMRT_TimerRunFlag (*C enumerator*), 535
drivers_RT_RT600_MIMXRT685S._mrt_timer_mode (*C enum*), 534
drivers_RT_RT600_MIMXRT685S._mrt_timer_mode.kMRT_OneShotMode (*C enumerator*), 534
drivers_RT_RT600_MIMXRT685S._mrt_timer_mode.kMRT_OneShotStallMode (*C enumerator*), 534
drivers_RT_RT600_MIMXRT685S._mrt_timer_mode.kMRT_RepeatMode (*C enumerator*), 534
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable (*C enum*), 543

drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_GenInt0InterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_GenInt1InterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_GenInt2InterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_GenInt3InterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_HardwareResetInterruptEnable (*C enumerator*), 544
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_MuResetInterruptEnable (*C enumerator*), 544
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_ResetAssertInterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_ResetDeassertInterruptEnable (*C enumerator*), 544
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_Rx0FullInterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_Rx1FullInterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_Rx2FullInterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_Rx3FullInterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_Tx0EmptyInterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_Tx1EmptyInterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_Tx2EmptyInterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_enable.kMU_Tx3EmptyInterruptEnable (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_interrupt_trigger (*C enum*), 544
drivers_RT_RT600_MIMXRT685S._mu_interrupt_trigger.kMU_GenInt0InterruptTrigger (*C enumerator*), 544
drivers_RT_RT600_MIMXRT685S._mu_interrupt_trigger.kMU_GenInt1InterruptTrigger (*C enumerator*), 544
drivers_RT_RT600_MIMXRT685S._mu_interrupt_trigger.kMU_GenInt2InterruptTrigger (*C enumerator*), 544
drivers_RT_RT600_MIMXRT685S._mu_interrupt_trigger.kMU_GenInt3InterruptTrigger (*C enumerator*), 544
drivers_RT_RT600_MIMXRT685S._mu_msg_reg_index (*C enum*), 544
drivers_RT_RT600_MIMXRT685S._mu_msg_reg_index.kMU_MsgReg0 (*C enumerator*), 544
drivers_RT_RT600_MIMXRT685S._mu_msg_reg_index.kMU_MsgReg1 (*C enumerator*), 544
drivers_RT_RT600_MIMXRT685S._mu_msg_reg_index.kMU_MsgReg2 (*C enumerator*), 544
drivers_RT_RT600_MIMXRT685S._mu_msg_reg_index.kMU_MsgReg3 (*C enumerator*), 544
drivers_RT_RT600_MIMXRT685S._mu_status_flags (*C enum*), 541
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_EventPendingFlag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_FlagsUpdatingFlag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_GenInt0Flag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_GenInt1Flag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_GenInt2Flag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_GenInt3Flag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_HardwareResetInterruptFlag (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_MuResetInterruptFlag (*C enumerator*), 543
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_OtherSideInResetFlag (*C enumerator*),

542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_ResetAssertInterruptFlag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_ResetDeassertInterruptFlag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_Rx0FullFlag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_Rx1FullFlag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_Rx2FullFlag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_Rx3FullFlag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_Tx0EmptyFlag (*C enumerator*), 541
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_Tx1EmptyFlag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_Tx2EmptyFlag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._mu_status_flags.kMU_Tx3EmptyFlag (*C enumerator*), 542
drivers_RT_RT600_MIMXRT685S._operation_mode (*C enum*), 304
drivers_RT_RT600_MIMXRT685S._operation_mode.kDMIC_OperationModeDma (*C enumerator*), 304
drivers_RT_RT600_MIMXRT685S._operation_mode.kDMIC_OperationModeInterrupt (*C enumerator*), 304
drivers_RT_RT600_MIMXRT685S._ostimer_flags (*C enum*), 550
drivers_RT_RT600_MIMXRT685S._ostimer_flags.kOSTIMER_MatchInterruptFlag (*C enumerator*), 550
drivers_RT_RT600_MIMXRT685S._otfad_config (*C struct*), 553
drivers_RT_RT600_MIMXRT685S._otfad_config.enableIntRequest (*C var*), 553
drivers_RT_RT600_MIMXRT685S._otfad_config.enableOTFAD (*C var*), 553
drivers_RT_RT600_MIMXRT685S._otfad_config.forceError (*C var*), 553
drivers_RT_RT600_MIMXRT685S._otfad_config.forceLDM (*C var*), 553
drivers_RT_RT600_MIMXRT685S._otfad_config.forceSVM (*C var*), 553
drivers_RT_RT600_MIMXRT685S._otfad_config.keyBlobProcess (*C var*), 553
drivers_RT_RT600_MIMXRT685S._otfad_config.keyBlobScramble (*C var*), 553
drivers_RT_RT600_MIMXRT685S._otfad_config.restrictedRegAccess (*C var*), 553
drivers_RT_RT600_MIMXRT685S._otfad_config.startKeyBlobProcessing (*C var*), 553
drivers_RT_RT600_MIMXRT685S._otfad_encryption_config (*C struct*), 552
drivers_RT_RT600_MIMXRT685S._otfad_encryption_config.AESdecryption (*C var*), 552
drivers_RT_RT600_MIMXRT685S._otfad_encryption_config.contextIndex (*C var*), 552
drivers_RT_RT600_MIMXRT685S._otfad_encryption_config.counter (*C var*), 553
drivers_RT_RT600_MIMXRT685S._otfad_encryption_config.endAddr (*C var*), 552
drivers_RT_RT600_MIMXRT685S._otfad_encryption_config.key (*C var*), 553
drivers_RT_RT600_MIMXRT685S._otfad_encryption_config.readOnly (*C var*), 552
drivers_RT_RT600_MIMXRT685S._otfad_encryption_config.startAddr (*C var*), 552
drivers_RT_RT600_MIMXRT685S._otfad_encryption_config.valid (*C var*), 552
drivers_RT_RT600_MIMXRT685S._pint_int (*C enum*), 554
drivers_RT_RT600_MIMXRT685S._pint_int.kPINT_PinInt0 (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pin_enable (*C enum*), 553
drivers_RT_RT600_MIMXRT685S._pint_pin_enable.kPINT_PinIntEnableBothEdges (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pin_enable.kPINT_PinIntEnableFallEdge (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pin_enable.kPINT_PinIntEnableHighLevel (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pin_enable.kPINT_PinIntEnableLowLevel (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pin_enable.kPINT_PinIntEnableNone (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pin_enable.kPINT_PinIntEnableRiseEdge (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pmatch_bslice (*C enum*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_bslice_cfg (*C enum*), 555

drivers_RT_RT600_MIMXRT685S._pint_pmatch_bslice_cfg.kPINT_PatternMatchAlways (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_bslice_cfg.kPINT_PatternMatchBothEdges (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_bslice_cfg.kPINT_PatternMatchHigh (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_bslice_cfg.kPINT_PatternMatchLow (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_bslice_cfg.kPINT_PatternMatchNever (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_bslice_cfg.kPINT_PatternMatchStickyBothEdges (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_bslice_cfg.kPINT_PatternMatchStickyFall (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_bslice_cfg.kPINT_PatternMatchStickyRise (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_bslice.kPINT_PatternMatchBSlice0 (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_cfg (*C struct*), 563
drivers_RT_RT600_MIMXRT685S._pint_pmatch_input_src (*C enum*), 554
drivers_RT_RT600_MIMXRT685S._pint_pmatch_input_src.kPINT_PatternMatchInp0Src (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pmatch_input_src.kPINT_PatternMatchInp1Src (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pmatch_input_src.kPINT_PatternMatchInp2Src (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pmatch_input_src.kPINT_PatternMatchInp3Src (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pmatch_input_src.kPINT_PatternMatchInp4Src (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pmatch_input_src.kPINT_PatternMatchInp5Src (*C enumerator*), 554
drivers_RT_RT600_MIMXRT685S._pint_pmatch_input_src.kPINT_PatternMatchInp6Src (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_input_src.kPINT_PatternMatchInp7Src (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_input_src.kPINT_SecPatternMatchInp0Src (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_pmatch_input_src.kPINT_SecPatternMatchInp1Src (*C enumerator*), 555
drivers_RT_RT600_MIMXRT685S._pint_status (*C struct*), 563
drivers_RT_RT600_MIMXRT685S._pmc_event_flags (*C enum*), 564
drivers_RT_RT600_MIMXRT685S._pmc_event_flags.kPMC_FLAGS_AUTOWK (*C enumerator*), 564
drivers_RT_RT600_MIMXRT685S._pmc_event_flags.kPMC_FLAGS_DEEPPD (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S._pmc_event_flags.kPMC_FLAGS_HVD1V8 (*C enumerator*), 564
drivers_RT_RT600_MIMXRT685S._pmc_event_flags.kPMC_FLAGS_HVDCORE (*C enumerator*), 564
drivers_RT_RT600_MIMXRT685S._pmc_event_flags.kPMC_FLAGS_INTNPADF (*C enumerator*), 564
drivers_RT_RT600_MIMXRT685S._pmc_event_flags.kPMC_FLAGS_LVDCORE (*C enumerator*), 564
drivers_RT_RT600_MIMXRT685S._pmc_event_flags.kPMC_FLAGS_POR1V8 (*C enumerator*), 564
drivers_RT_RT600_MIMXRT685S._pmc_event_flags.kPMC_FLAGS_PORAO18 (*C enumerator*), 564

drivers_RT_RT600_MIMXRT685S._pmc_event_flags.kPMC_FLAGS_PORCORE (*C enumerator*), 564
drivers_RT_RT600_MIMXRT685S._pmc_event_flags.kPMC_FLAGS_RESETNPAD (*C enumerator*), 564
drivers_RT_RT600_MIMXRT685S._pmc_event_flags.kPMC_FLAGS_RTC (*C enumerator*), 564
drivers_RT_RT600_MIMXRT685S._pmc_interrupt (*C enum*), 563
drivers_RT_RT600_MIMXRT685S._pmc_interrupt.kPMC_INT_AUTOWK (*C enumerator*), 564
drivers_RT_RT600_MIMXRT685S._pmc_interrupt.kPMC_INT_HVD1V8 (*C enumerator*), 564
drivers_RT_RT600_MIMXRT685S._pmc_interrupt.kPMC_INT_HVDCORE (*C enumerator*), 563
drivers_RT_RT600_MIMXRT685S._pmc_interrupt.kPMC_INT_INTRPAD (*C enumerator*), 564
drivers_RT_RT600_MIMXRT685S._pmc_interrupt.kPMC_INT_LVDCORE (*C enumerator*), 563
drivers_RT_RT600_MIMXRT685S._pmic_mode_reg (*C enum*), 570
drivers_RT_RT600_MIMXRT685S._pmic_mode_reg.kCfg_Run (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._pmic_mode_reg.kCfg_Sleep (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._power_control_for_pmic_mode (*C enum*), 572
drivers_RT_RT600_MIMXRT685S._power_control_for_pmic_mode.kVdd1v8OnMode1 (*C enumerator*), 572
drivers_RT_RT600_MIMXRT685S._power_control_for_pmic_mode.kVdd1v8OnMode2 (*C enumerator*), 572
drivers_RT_RT600_MIMXRT685S._power_control_for_pmic_mode.kVdd1v8OnMode3 (*C enumerator*), 572
drivers_RT_RT600_MIMXRT685S._power_control_for_pmic_mode.kVddCoreOnMode1 (*C enumerator*), 572
drivers_RT_RT600_MIMXRT685S._power_control_for_pmic_mode.kVddCoreOnMode2 (*C enumerator*), 572
drivers_RT_RT600_MIMXRT685S._power_control_for_pmic_mode.kVddCoreOnMode3 (*C enumerator*), 572
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val (*C enum*), 570
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_720 (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_735 (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_750 (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_765 (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_780 (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_795 (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_810 (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_825 (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_840 (*C enumerator*), 571
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_855 (*C enumerator*), 571
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_870 (*C enumerator*), 571
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_885 (*C enumerator*), 571
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_900 (*C enumerator*), 571
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_915 (*C enumerator*), 571
drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_930 (*C enumerator*), 571

drivers_RT_RT600_MIMXRT685S._power_lvd_falling_trip_vol_val.kLvdFallingTripVol_945 (*C enumerator*), 571
drivers_RT_RT600_MIMXRT685S._power_mode_config (*C enum*), 569
drivers_RT_RT600_MIMXRT685S._power_mode_config.kPmu_Deep_PowerDown (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S._power_mode_config.kPmu_Deep_Sleep (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S._power_mode_config.kPmu_Full_Deep_PowerDown (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S._power_mode_config.kPmu_Sleep (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S._power_pad_vrange (*C struct*), 578
drivers_RT_RT600_MIMXRT685S._power_pad_vrange_val (*C enum*), 570
drivers_RT_RT600_MIMXRT685S._power_pad_vrange_val.kPadVol_171_198 (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._power_pad_vrange_val.kPadVol_171_360 (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._power_pad_vrange_val.kPadVol_300_360 (*C enumerator*), 570
drivers_RT_RT600_MIMXRT685S._power_part_temp_range (*C enum*), 571
drivers_RT_RT600_MIMXRT685S._power_part_temp_range.kPartTemp_0C_P85C (*C enumerator*), 571
drivers_RT_RT600_MIMXRT685S._power_part_temp_range.kPartTemp_N20C_P85C (*C enumerator*), 571
drivers_RT_RT600_MIMXRT685S._power_vddcore_src (*C enum*), 571
drivers_RT_RT600_MIMXRT685S._power_vddcore_src.kVddCoreSrc_LDO (*C enumerator*), 572
drivers_RT_RT600_MIMXRT685S._power_vddcore_src.kVddCoreSrc_PMIC (*C enumerator*), 572
drivers_RT_RT600_MIMXRT685S._power_volt_op_range (*C enum*), 571
drivers_RT_RT600_MIMXRT685S._power_volt_op_range.kVoltOpFullRange (*C enumerator*), 571
drivers_RT_RT600_MIMXRT685S._power_volt_op_range.kVoltOpLowRange (*C enumerator*), 571
drivers_RT_RT600_MIMXRT685S._pq_biquad0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_biquad0_fx (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_biquad1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_biquad1_fx (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_biquad_param (*C struct*), 611
drivers_RT_RT600_MIMXRT685S._pq_biquad_param.a_1 (*C var*), 611
drivers_RT_RT600_MIMXRT685S._pq_biquad_param.a_2 (*C var*), 611
drivers_RT_RT600_MIMXRT685S._pq_biquad_param.b_0 (*C var*), 611
drivers_RT_RT600_MIMXRT685S._pq_biquad_param.b_1 (*C var*), 611
drivers_RT_RT600_MIMXRT685S._pq_biquad_param.b_2 (*C var*), 611
drivers_RT_RT600_MIMXRT685S._pq_biquad_param.v_n (*C var*), 611
drivers_RT_RT600_MIMXRT685S._pq_biquad_param.v_n_1 (*C var*), 611
drivers_RT_RT600_MIMXRT685S._pq_biquad_state (*C struct*), 611
drivers_RT_RT600_MIMXRT685S._pq_biquad_state.compreg (*C var*), 612
drivers_RT_RT600_MIMXRT685S._pq_biquad_state.param (*C var*), 612
drivers_RT_RT600_MIMXRT685S._pq_cos0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_cos1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_cos_fx0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_cos_fx1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_div0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_div1 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_etonx0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_etonx1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_etonx_fx0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_etonx_fx1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_etox0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_etox1 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_etox_fx0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_etox_fx1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_float (*C union*), 612
drivers_RT_RT600_MIMXRT685S._pq_float.floatX (*C var*), 612
drivers_RT_RT600_MIMXRT685S._pq_float.integerX (*C var*), 612
drivers_RT_RT600_MIMXRT685S._pq_inv0 (*C macro*), 598

drivers_RT_RT600_MIMXRT685S._pq_inv1 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_inv_fx0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_inv_fx1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_invsqrt0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_invsqrt1 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_invsqrt_fx0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_invsqrt_fx1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_ln0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_ln1 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_ln_fx0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_ln_fx1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_readAdd0 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_readAdd0_fx (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_readAdd1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_readAdd1_fx (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_readMult0 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_readMult0_fx (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_readMult1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_readMult1_fx (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_sin0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_sin1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_sin_fx0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_sin_fx1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._pq_sqrt0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_sqrt1 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_sqrt_fx0 (*C macro*), 598
drivers_RT_RT600_MIMXRT685S._pq_sqrt_fx1 (*C macro*), 599
drivers_RT_RT600_MIMXRT685S._puf_key_index_register (*C enum*), 613
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_00 (*C enumerator*), 613
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_01 (*C enumerator*), 613
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_02 (*C enumerator*), 613
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_03 (*C enumerator*), 613
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_04 (*C enumerator*), 613
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_05 (*C enumerator*), 613
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_06 (*C enumerator*), 613
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_07 (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_08 (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_09 (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_10 (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_11 (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_12 (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_13 (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_14 (*C enumerator*), 614

drivers_RT_RT600_MIMXRT685S._puf_key_index_register.kPUF_KeyIndex_15 (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._puf_key_slot (*C enum*), 614
drivers_RT_RT600_MIMXRT685S._puf_key_slot.kPUF_KeySlot0 (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._puf_key_slot.kPUF_KeySlot1 (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._puf_min_max (*C enum*), 614
drivers_RT_RT600_MIMXRT685S._puf_min_max.kPUF_KeyIndexMax (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._puf_min_max.kPUF_KeySizeMax (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._puf_min_max.kPUF_KeySizeMin (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn (*C enum*), 615
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kACMP0_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kADC0_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kCASPER_RST_SHIFT_RSTn (*C enumerator*), 615
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kCRC_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kCT32B0_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kCT32B1_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kCT32B2_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kCT32B3_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kCT32B4_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kDMAC0_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kDMAC1_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kDMIC_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kDSP_RST_SHIFT_RSTn (*C enumerator*), 615
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kFC0_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kFC1_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kFC2_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kFC3_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kFC4_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kFC5_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kFC6_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kFC7_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kFC14_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kFC15_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kFLEXSPI_RST_SHIFT_RSTn (*C enumerator*), 615

drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kFREQME_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kHASHCRYPT_RST_SHIFT_RSTn (*C enumerator*), 615
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kHSGPIO0_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kHSGPIO1_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kHSGPIO2_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kHSGPIO3_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kHSGPIO4_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kHSGPIO5_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kHSGPIO6_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kHSGPIO7_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kI3C0_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kINPUTMUX_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kMRT0_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kMU_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kOSEVENT_TIMER_RST_SHIFT_RSTn (*C enumerator*), 617
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kPINT_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kPOWERQUAD_RST_SHIFT_RSTn (*C enumerator*), 615
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kPUF_RST_SHIFT_RSTn (*C enumerator*), 615
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kRNG_RST_SHIFT_RSTn (*C enumerator*), 615
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kSCT_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kSDIO0_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kSDIO1_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kSEMA_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kSHSGPIO0_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kUSBHS_DEVICE_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kUSBHS_HOST_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kUSBHS_PHY_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kUSBHS_SRAM_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kUTICK0_RST_SHIFT_RSTn (*C enumerator*), 616

drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kWWDT0_RST_SHIFT_RSTn (*C enumerator*), 616
drivers_RT_RT600_MIMXRT685S._RSTCTL_RSTn.kWWDT1_RST_SHIFT_RSTn (*C enumerator*), 618
drivers_RT_RT600_MIMXRT685S._rtc_datetime (*C struct*), 627
drivers_RT_RT600_MIMXRT685S._rtc_datetime.day (*C var*), 627
drivers_RT_RT600_MIMXRT685S._rtc_datetime.hour (*C var*), 627
drivers_RT_RT600_MIMXRT685S._rtc_datetime.minute (*C var*), 627
drivers_RT_RT600_MIMXRT685S._rtc_datetime.month (*C var*), 627
drivers_RT_RT600_MIMXRT685S._rtc_datetime.second (*C var*), 627
drivers_RT_RT600_MIMXRT685S._rtc_datetime.year (*C var*), 627
drivers_RT_RT600_MIMXRT685S._rtc_interrupt_enable (*C enum*), 625
drivers_RT_RT600_MIMXRT685S._rtc_interrupt_enable.kRTC_AlarmInterruptEnable (*C enumerator*), 625
drivers_RT_RT600_MIMXRT685S._rtc_interrupt_enable.kRTC_WakeupInterruptEnable (*C enumerator*), 625
drivers_RT_RT600_MIMXRT685S._rtc_status_flags (*C enum*), 625
drivers_RT_RT600_MIMXRT685S._rtc_status_flags.kRTC_AlarmFlag (*C enumerator*), 625
drivers_RT_RT600_MIMXRT685S._rtc_status_flags.kRTC_WakeupFlag (*C enumerator*), 625
drivers_RT_RT600_MIMXRT685S._sctimer_clock_mode (*C enum*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_clock_mode.kSCTIMER_Asynchronous_ClockMode (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_mode.kSCTIMER_Input_ClockMode (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_mode.kSCTIMER_Sampled_ClockMode (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_mode.kSCTIMER_System_ClockMode (*C enumerator*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select (*C enum*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_0 (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_1 (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_2 (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_3 (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_4 (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_5 (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_6 (*C enumerator*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Fall_Input_7 (*C enumerator*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_0 (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_1 (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_2 (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_3 (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_4 (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_5 (*C enumerator*), 643
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_6 (*C*

enumerator), 644
drivers_RT_RT600_MIMXRT685S._sctimer_clock_select.kSCTIMER_Clock_On_Rise_Input_7 (*C enumerator*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_complementary_pwm_handle (*C struct*), 649
drivers_RT_RT600_MIMXRT685S._sctimer_complementary_pwm_handle.deadTimeTicks (*C var*), 650
drivers_RT_RT600_MIMXRT685S._sctimer_complementary_pwm_handle.highFallEvent (*C var*), 650
drivers_RT_RT600_MIMXRT685S._sctimer_complementary_pwm_handle.highRiseEvent (*C var*), 650
drivers_RT_RT600_MIMXRT685S._sctimer_complementary_pwm_handle.lowRiseEvent (*C var*), 650
drivers_RT_RT600_MIMXRT685S._sctimer_complementary_pwm_handle.outHigh (*C var*), 650
drivers_RT_RT600_MIMXRT685S._sctimer_complementary_pwm_handle.outLow (*C var*), 650
drivers_RT_RT600_MIMXRT685S._sctimer_complementary_pwm_handle.periodEvent (*C var*), 649
drivers_RT_RT600_MIMXRT685S._sctimer_config (*C struct*), 650
drivers_RT_RT600_MIMXRT685S._sctimer_config.clockMode (*C var*), 650
drivers_RT_RT600_MIMXRT685S._sctimer_config.clockSelect (*C var*), 650
drivers_RT_RT600_MIMXRT685S._sctimer_config.enableBidirection_h (*C var*), 651
drivers_RT_RT600_MIMXRT685S._sctimer_config.enableBidirection_l (*C var*), 650
drivers_RT_RT600_MIMXRT685S._sctimer_config.enableCounterUnify (*C var*), 650
drivers_RT_RT600_MIMXRT685S._sctimer_config.inputsync (*C var*), 651
drivers_RT_RT600_MIMXRT685S._sctimer_config.outInitState (*C var*), 651
drivers_RT_RT600_MIMXRT685S._sctimer_config.prescale_h (*C var*), 651
drivers_RT_RT600_MIMXRT685S._sctimer_config.prescale_l (*C var*), 651
drivers_RT_RT600_MIMXRT685S._sctimer_conflict_resolution (*C enum*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_conflict_resolution.kSCTIMER_ResolveClear (*C enumerator*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_conflict_resolution.kSCTIMER_ResolveNone (*C enumerator*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_conflict_resolution.kSCTIMER_ResolveSet (*C enumerator*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_conflict_resolution.kSCTIMER_ResolveToggle (*C enumerator*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_counter (*C enum*), 640
drivers_RT_RT600_MIMXRT685S._sctimer_counter.kSCTIMER_Counter_H (*C enumerator*), 640
drivers_RT_RT600_MIMXRT685S._sctimer_counter.kSCTIMER_Counter_L (*C enumerator*), 640
drivers_RT_RT600_MIMXRT685S._sctimer_counter.kSCTIMER_Counter_U (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_event (*C enum*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_event_active_direction (*C enum*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_event_active_direction.kSCTIMER_ActiveInCountDown (*C enumerator*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_event_active_direction.kSCTIMER_ActiveInCountUp (*C enumerator*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_event_active_direction.kSCTIMER_ActiveIndependent (*C enumerator*), 644
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_InputFallAndMatchEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_InputFallEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_InputFallOrMatchEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_InputHighAndMatchEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_InputHighEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_InputHighOrMatchEvent (*C enumerator*), 645

drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_InputLowAndMatchEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_InputLowEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_InputLowOrMatchEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_InputRiseAndMatchEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_InputRiseEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_InputRiseOrMatchEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_MatchEventOnly (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_OutputFallAndMatchEvent (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_OutputFallEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_OutputFallOrMatchEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_OutputHighAndMatchEvent (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_OutputHighEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_OutputHighOrMatchEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_OutputLowAndMatchEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_OutputLowEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_OutputLowOrMatchEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_OutputRiseAndMatchEvent (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_OutputRiseEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_event.kSCTIMER_OutputRiseOrMatchEvent (*C enumerator*), 645
drivers_RT_RT600_MIMXRT685S._sctimer_input (*C enum*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_input.kSCTIMER_Input_0 (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_input.kSCTIMER_Input_1 (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_input.kSCTIMER_Input_2 (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_input.kSCTIMER_Input_3 (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_input.kSCTIMER_Input_4 (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_input.kSCTIMER_Input_5 (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_input.kSCTIMER_Input_6 (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_input.kSCTIMER_Input_7 (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable (*C enum*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event0InterruptEnable (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event1InterruptEnable (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event2InterruptEnable (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event3InterruptEnable (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event4InterruptEnable (*C enumerator*), 646

drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event5InterruptEnable (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event6InterruptEnable (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event7InterruptEnable (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event8InterruptEnable (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event9InterruptEnable (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event10InterruptEnable (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event11InterruptEnable (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_interrupt_enable.kSCTIMER_Event12InterruptEnable (*C enumerator*), 646
drivers_RT_RT600_MIMXRT685S._sctimer_out (*C enum*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_out.kSCTIMER_Out_0 (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_out.kSCTIMER_Out_1 (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_out.kSCTIMER_Out_2 (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_out.kSCTIMER_Out_3 (*C enumerator*), 641
drivers_RT_RT600_MIMXRT685S._sctimer_out.kSCTIMER_Out_4 (*C enumerator*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_out.kSCTIMER_Out_5 (*C enumerator*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_out.kSCTIMER_Out_6 (*C enumerator*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_out.kSCTIMER_Out_7 (*C enumerator*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_out.kSCTIMER_Out_8 (*C enumerator*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_out.kSCTIMER_Out_9 (*C enumerator*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_level_select (*C enum*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_level_select.kSCTIMER_HighTrue (*C enumerator*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_level_select.kSCTIMER_LowTrue (*C enumerator*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_mode (*C enum*), 640
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_mode.kSCTIMER_CenterAlignedPwm (*C enumerator*), 640
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_mode.kSCTIMER_EdgeAlignedPwm (*C enumerator*), 640
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_signal_param (*C struct*), 649
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_signal_param.dutyCyclePercent (*C var*), 649
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_signal_param.level (*C var*), 649
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_signal_param.output (*C var*), 649
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_update_mode (*C enum*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_update_mode.kSCTIMER_UpdateImmediately (*C enumerator*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_pwm_update_mode.kSCTIMER_UpdateOnNextPeriod (*C enumerator*), 642
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags (*C enum*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_BusErrorHFlag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_BusErrorLFlag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event0Flag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event1Flag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event2Flag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event3Flag (*C enumerator*),

647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event4Flag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event5Flag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event6Flag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event7Flag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event8Flag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event9Flag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event10Flag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event11Flag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._sctimer_status_flags.kSCTIMER_Event12Flag (*C enumerator*), 647
drivers_RT_RT600_MIMXRT685S._SDK_AtomicLocalCompareAndSet (*C function*), 504
drivers_RT_RT600_MIMXRT685S._SDK_AtomicTestAndSet (*C function*), 504
drivers_RT_RT600_MIMXRT685S._sema42_gate_status (*C enum*), 651
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc0 (*C enumerator*), 651
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc1 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc2 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc3 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc4 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc5 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc6 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc7 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc8 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc9 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc10 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc11 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc12 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc13 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_LockedByProc14 (*C enumerator*), 652
drivers_RT_RT600_MIMXRT685S._sema42_gate_status.kSEMA42_Unlocked (*C enumerator*), 651
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option (*C struct*), 454
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option0 (*C union*), 459
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option0.B (*C struct*), 459
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option0.B (*C var*), 459
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option0.B.cmd_pads (*C var*), 459

drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option0.B.device_type (*C var*), 459
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option0.B.max_freq (*C var*), 459
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option0.B.misc_mode (*C var*), 459
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option0.B.option_size (*C var*), 459
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option0.B.quad_mode_setting (*C var*), 459
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option0.B.query_pads (*C var*), 459
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option0.B.tag (*C var*), 459
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option0.U (*C var*), 459
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option1 (*C union*), 459
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option1.B (*C struct*), 460
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option1.B (*C var*), 460
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option1.B.dqs_pinmux_group (*C var*), 460
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option1.B.drive_strength (*C var*), 460
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option1.B.dummy_cycles (*C var*), 460
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option1.B.flash_connection (*C var*), 460
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option1.B.pinmux_group (*C var*), 460
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option1.B.status_override (*C var*), 460
drivers_RT_RT600_MIMXRT685S._serial_nor_config_option.option1.U (*C var*), 460
drivers_RT_RT600_MIMXRT685S._spi_clock_phase (*C enum*), 660
drivers_RT_RT600_MIMXRT685S._spi_clock_phase.kSPI_ClockPhaseFirstEdge (*C enumerator*), 660
drivers_RT_RT600_MIMXRT685S._spi_clock_phase.kSPI_ClockPhaseSecondEdge (*C enumerator*), 660
drivers_RT_RT600_MIMXRT685S._spi_clock_polarity (*C enum*), 660
drivers_RT_RT600_MIMXRT685S._spi_clock_polarity.kSPI_ClockPolarityActiveHigh (*C enumerator*), 660
drivers_RT_RT600_MIMXRT685S._spi_clock_polarity.kSPI_ClockPolarityActiveLow (*C enumerator*), 660
drivers_RT_RT600_MIMXRT685S._spi_config (*C struct*), 668
drivers_RT_RT600_MIMXRT685S._spi_data_width (*C enum*), 661
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data4Bits (*C enumerator*), 661
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data5Bits (*C enumerator*), 661
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data6Bits (*C enumerator*), 661
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data7Bits (*C enumerator*), 661
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data8Bits (*C enumerator*), 661
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data9Bits (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data10Bits (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data11Bits (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data12Bits (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data13Bits (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data14Bits (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data15Bits (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_data_width.kSPI_Data16Bits (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_delay_config (*C struct*), 665
drivers_RT_RT600_MIMXRT685S._spi_delay_config.frameDelay (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_delay_config.postDelay (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_delay_config.preDelay (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_delay_config.transferDelay (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_dma_handle (*C struct*), 658
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.base (*C var*), 658
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.bytesPerFrame (*C var*), 658
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.callback (*C var*), 658
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.instance (*C var*), 659
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.isSlave (*C var*), 659
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.lastword (*C var*), 658

drivers_RT_RT600_MIMXRT685S._spi_dma_handle.lastwordBytes (*C var*), 658
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.rxHandle (*C var*), 658
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.rxInProgress (*C var*), 658
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.rxNextData (*C var*), 659
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.rxRemainingBytes (*C var*), 659
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.state (*C var*), 658
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.transferSize (*C var*), 658
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.txDummy (*C var*), 658
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.txHandle (*C var*), 658
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.txInProgress (*C var*), 658
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.txNextData (*C var*), 659
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.txRemainingBytes (*C var*), 659
drivers_RT_RT600_MIMXRT685S._spi_dma_handle.userData (*C var*), 658
drivers_RT_RT600_MIMXRT685S._spi_half_duplex_transfer (*C struct*), 668
drivers_RT_RT600_MIMXRT685S._spi_half_duplex_transfer.configFlags (*C var*), 668
drivers_RT_RT600_MIMXRT685S._spi_half_duplex_transfer.isPcsAssertInTransfer (*C var*), 668
drivers_RT_RT600_MIMXRT685S._spi_half_duplex_transfer.isTransmitFirst (*C var*), 668
drivers_RT_RT600_MIMXRT685S._spi_half_duplex_transfer.rxData (*C var*), 668
drivers_RT_RT600_MIMXRT685S._spi_half_duplex_transfer.rxDataSize (*C var*), 668
drivers_RT_RT600_MIMXRT685S._spi_half_duplex_transfer.txData (*C var*), 668
drivers_RT_RT600_MIMXRT685S._spi_half_duplex_transfer.txDataSize (*C var*), 668
drivers_RT_RT600_MIMXRT685S._spi_interrupt_enable (*C enum*), 663
drivers_RT_RT600_MIMXRT685S._spi_interrupt_enable.kSPI_RxLvlIrq (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S._spi_interrupt_enable.kSPI_TxLvlIrq (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S._spi_master_config (*C struct*), 666
drivers_RT_RT600_MIMXRT685S._spi_master_config.baudRate_Bps (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_master_config.dataWidth (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_master_config.delayConfig (*C var*), 667
drivers_RT_RT600_MIMXRT685S._spi_master_config.direction (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_master_config.enableLoopback (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_master_config.enableMaster (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_master_config.phase (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_master_config.polarity (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_master_config.rxWatermark (*C var*), 667
drivers_RT_RT600_MIMXRT685S._spi_master_config.sselNum (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_master_config.sselPol (*C var*), 666
drivers_RT_RT600_MIMXRT685S._spi_master_config.txWatermark (*C var*), 667
drivers_RT_RT600_MIMXRT685S._spi_master_handle (*C struct*), 668
drivers_RT_RT600_MIMXRT685S._spi_master_handle.callback (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.dataWidth (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.lastControl (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.rxData (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.rxRemainingBytes (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.rxWatermark (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.state (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.toReceiveCount (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.totalByteCount (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.txControl (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.txData (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.txRemainingBytes (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.txWatermark (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_master_handle.userData (*C var*), 669
drivers_RT_RT600_MIMXRT685S._spi_rxfifo_watermark (*C enum*), 661
drivers_RT_RT600_MIMXRT685S._spi_rxfifo_watermark.kSPI_RxFifo1 (*C enumerator*), 661
drivers_RT_RT600_MIMXRT685S._spi_rxfifo_watermark.kSPI_RxFifo2 (*C enumerator*), 661
drivers_RT_RT600_MIMXRT685S._spi_rxfifo_watermark.kSPI_RxFifo3 (*C enumerator*), 661
drivers_RT_RT600_MIMXRT685S._spi_rxfifo_watermark.kSPI_RxFifo4 (*C enumerator*), 661
drivers_RT_RT600_MIMXRT685S._spi_rxfifo_watermark.kSPI_RxFifo5 (*C enumerator*), 661

drivers_RT_RT600_MIMXRT685S._spi_rxfifo_watermark.kSPI_RxFifo6 (*C enumerator*), 661
drivers_RT_RT600_MIMXRT685S._spi_rxfifo_watermark.kSPI_RxFifo7 (*C enumerator*), 661
drivers_RT_RT600_MIMXRT685S._spi_rxfifo_watermark.kSPI_RxFifo8 (*C enumerator*), 661
drivers_RT_RT600_MIMXRT685S._spi_shift_direction (*C enum*), 659
drivers_RT_RT600_MIMXRT685S._spi_shift_direction.kSPI_LsbFirst (*C enumerator*), 659
drivers_RT_RT600_MIMXRT685S._spi_shift_direction.kSPI_MsbFirst (*C enumerator*), 659
drivers_RT_RT600_MIMXRT685S._spi_slave_config (*C struct*), 667
drivers_RT_RT600_MIMXRT685S._spi_slave_config.dataWidth (*C var*), 667
drivers_RT_RT600_MIMXRT685S._spi_slave_config.direction (*C var*), 667
drivers_RT_RT600_MIMXRT685S._spi_slave_config.enableSlave (*C var*), 667
drivers_RT_RT600_MIMXRT685S._spi_slave_config.phase (*C var*), 667
drivers_RT_RT600_MIMXRT685S._spi_slave_config.polarity (*C var*), 667
drivers_RT_RT600_MIMXRT685S._spi_slave_config.rxWatermark (*C var*), 667
drivers_RT_RT600_MIMXRT685S._spi_slave_config.sselPol (*C var*), 667
drivers_RT_RT600_MIMXRT685S._spi_slave_config.txWatermark (*C var*), 667
drivers_RT_RT600_MIMXRT685S._spi_spol (*C enum*), 662
drivers_RT_RT600_MIMXRT685S._spi_spol.kSPI_Spol0ActiveHigh (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_spol.kSPI_Spol1ActiveHigh (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_spol.kSPI_Spol3ActiveHigh (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S._spi_spol.kSPI_SpolActiveAllHigh (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S._spi_spol.kSPI_SpolActiveAllLow (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S._spi_ssel (*C enum*), 662
drivers_RT_RT600_MIMXRT685S._spi_ssel.kSPI_Ssel0 (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_ssel.kSPI_Ssel1 (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_ssel.kSPI_Ssel2 (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_ssel.kSPI_Ssel3 (*C enumerator*), 662
drivers_RT_RT600_MIMXRT685S._spi_statusflags (*C enum*), 663
drivers_RT_RT600_MIMXRT685S._spi_statusflags.kSPI_RxFullFlag (*C enumerator*), 664
drivers_RT_RT600_MIMXRT685S._spi_statusflags.kSPI_RxNotEmptyFlag (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S._spi_statusflags.kSPI_TxEmptyFlag (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S._spi_statusflags.kSPI_TxNotFullFlag (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S._spi_transfer (*C struct*), 667
drivers_RT_RT600_MIMXRT685S._spi_transfer.configFlags (*C var*), 668
drivers_RT_RT600_MIMXRT685S._spi_transfer.dataSize (*C var*), 668
drivers_RT_RT600_MIMXRT685S._spi_transfer.rxData (*C var*), 668
drivers_RT_RT600_MIMXRT685S._spi_transfer.txData (*C var*), 667
drivers_RT_RT600_MIMXRT685S._spi_txfifo_watermark (*C enum*), 660
drivers_RT_RT600_MIMXRT685S._spi_txfifo_watermark.kSPI_TxFifo0 (*C enumerator*), 660
drivers_RT_RT600_MIMXRT685S._spi_txfifo_watermark.kSPI_TxFifo1 (*C enumerator*), 660
drivers_RT_RT600_MIMXRT685S._spi_txfifo_watermark.kSPI_TxFifo2 (*C enumerator*), 660
drivers_RT_RT600_MIMXRT685S._spi_txfifo_watermark.kSPI_TxFifo3 (*C enumerator*), 660
drivers_RT_RT600_MIMXRT685S._spi_txfifo_watermark.kSPI_TxFifo4 (*C enumerator*), 660
drivers_RT_RT600_MIMXRT685S._spi_txfifo_watermark.kSPI_TxFifo5 (*C enumerator*), 660
drivers_RT_RT600_MIMXRT685S._spi_txfifo_watermark.kSPI_TxFifo6 (*C enumerator*), 660
drivers_RT_RT600_MIMXRT685S._spi_txfifo_watermark.kSPI_TxFifo7 (*C enumerator*), 660
drivers_RT_RT600_MIMXRT685S._spi_xfer_option (*C enum*), 659
drivers_RT_RT600_MIMXRT685S._spi_xfer_option.kSPI_FrameAssert (*C enumerator*), 659
drivers_RT_RT600_MIMXRT685S._spi_xfer_option.kSPI_FrameAssertEachFrame (*C enumerator*), 659
drivers_RT_RT600_MIMXRT685S._spi_xfer_option.kSPI_FrameDelay (*C enumerator*), 659
drivers_RT_RT600_MIMXRT685S._status_groups (*C enum*), 493
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_AON_COMMON (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_AON_POWER (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_ApplicationRangeStart (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_ASRC (*C enumerator*), 500

drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_AUDMIX (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_BMA (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_BUTTON (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_CAAM (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_CLIF (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_CODEC (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_COMMON_TASK (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_CSI (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_CSS_PKC (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_DCP (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_DebugConsole (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_DMA (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_DMAMGR (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_DMIC (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_DSPI (*C enumerator*), 493
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_ECSPI (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_EDMA (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_ELA_CSEC (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_ELE (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_ELEMU (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_ELS_PKC (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_ENDAT3 (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_ENET (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_ENET_QOS (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_ESAI (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_EVENT_CTRL (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_EXTERN_EEPROM (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLASH (*C enumerator*), 493
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLASHIAP (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLEXCAN (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLEXCOMM_I2C (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLEXIO_A_FORMAT (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLEXIO_CAMERA (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLEXIO_I2C (*C enumerator*), 493
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLEXIO_I2S (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLEXIO_MCULCD (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLEXIO_SPI (*C enumerator*), 493
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLEXIO_T_FORMAT (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLEXIO_UART (*C enumerator*), 493
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_FLEXSPI (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_Generic (*C enumerator*), 493
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_GLIKEY (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HAL_ADC_SENSOR (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HAL_FLASH (*C enumerator*),

498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HAL_GPIO (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HAL_I2C (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HAL_I2S (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HAL_PWM (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HAL_RNG (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HAL_SPI (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HAL_TIMER (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HAL_UART (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HASHCRYPT (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HIPERFACE (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_HOSTIF (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_I2C (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_I2S (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_I3C (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_I3CBUS (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_IAP (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_ICS (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_IPED (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_IUART (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_JPEGDEC (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LED (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LIST (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LMEM (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LOG (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LPC_I2C (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LPC_I2C_1 (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LPC_MINISPI (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LPC_QSPI (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LPC_SPI (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LPC_SPI_SSP (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LPC_USART (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LPI2C (*C enumerator*), 493
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LPSCI (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LPSPI (*C enumerator*), 493
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LPUART (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_LTC (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_MCAN (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_MCG (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_MECC (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_MEM_MANAGER (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_MIPI_DSI (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_MMDC (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_MSCAN (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_MSG (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_NETC (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_NOTIFIER (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_NPX (*C enumerator*), 501
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_OSA (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_OTFAD (*C enumerator*), 500

drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_OTP (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_PDM (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_PHY (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_PNGDEC (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_POWER (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_POWER_MANAGER (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_PUF (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_QSCI (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_QSPI (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_QUEUEDSPI (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SAI (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SCG (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SDHC (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SDIF (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SDIOSLV (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SDK_FLEXSPINOR (*C enumerator*), 500
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SDK_OCOTP (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SDMA (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SDMMC (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SDRAMC (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SDSPI (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SEMA42 (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SEMC (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SERIALMANAGER (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SFA (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SHELL (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SMARTCARD (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SPC (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SPDIF (*C enumerator*), 497
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SPI (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_SPIFI (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_TIMERMANAGER (*C enumerator*), 499
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_TOUCH_PANEL (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_TRGMUX (*C enumerator*), 495
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_UART (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_USDHC (*C enumerator*), 496
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_VBAT (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_XRDC (*C enumerator*), 494
drivers_RT_RT600_MIMXRT685S._status_groups.kStatusGroup_XSPI (*C enumerator*), 498
drivers_RT_RT600_MIMXRT685S._stereo_side (*C enum*), 304
drivers_RT_RT600_MIMXRT685S._stereo_side.kDMIC_Left (*C enumerator*), 304
drivers_RT_RT600_MIMXRT685S._stereo_side.kDMIC_Right (*C enumerator*), 304
drivers_RT_RT600_MIMXRT685S._sys_pll_mult (*C enum*), 248
drivers_RT_RT600_MIMXRT685S._sys_pll_mult.kCLOCK_SysPllMult16 (*C enumerator*), 248
drivers_RT_RT600_MIMXRT685S._sys_pll_mult.kCLOCK_SysPllMult17 (*C enumerator*), 249
drivers_RT_RT600_MIMXRT685S._sys_pll_mult.kCLOCK_SysPllMult18 (*C enumerator*), 249
drivers_RT_RT600_MIMXRT685S._sys_pll_mult.kCLOCK_SysPllMult19 (*C enumerator*), 249
drivers_RT_RT600_MIMXRT685S._sys_pll_mult.kCLOCK_SysPllMult20 (*C enumerator*), 249
drivers_RT_RT600_MIMXRT685S._sys_pll_mult.kCLOCK_SysPllMult21 (*C enumerator*), 249
drivers_RT_RT600_MIMXRT685S._sys_pll_mult.kCLOCK_SysPllMult22 (*C enumerator*), 249

drivers_RT_RT600_MIMXRT685S._sys_pll_src (*C enum*), 248
drivers_RT_RT600_MIMXRT685S._sys_pll_src.kCLOCK_SysPllFFroDiv2 (*C enumerator*), 248
drivers_RT_RT600_MIMXRT685S._sys_pll_src.kCLOCK_SysPllNone (*C enumerator*), 248
drivers_RT_RT600_MIMXRT685S._sys_pll_src.kCLOCK_SysPllSFroClk (*C enumerator*), 248
drivers_RT_RT600_MIMXRT685S._sys_pll_src.kCLOCK_SysPllXtalIn (*C enumerator*), 248
drivers_RT_RT600_MIMXRT685S._trng_clock_mode (*C enum*), 671
drivers_RT_RT600_MIMXRT685S._trng_clock_mode.kTRNG_ClockModeRingOscillator (*C enumerator*), 671
drivers_RT_RT600_MIMXRT685S._trng_clock_mode.kTRNG_ClockModeSystem (*C enumerator*), 671
drivers_RT_RT600_MIMXRT685S._trng_ring_osc_div (*C enum*), 671
drivers_RT_RT600_MIMXRT685S._trng_ring_osc_div.kTRNG_RingOscDiv0 (*C enumerator*), 672
drivers_RT_RT600_MIMXRT685S._trng_ring_osc_div.kTRNG_RingOscDiv2 (*C enumerator*), 672
drivers_RT_RT600_MIMXRT685S._trng_ring_osc_div.kTRNG_RingOscDiv4 (*C enumerator*), 672
drivers_RT_RT600_MIMXRT685S._trng_ring_osc_div.kTRNG_RingOscDiv8 (*C enumerator*), 672
drivers_RT_RT600_MIMXRT685S._trng_sample_mode (*C enum*), 671
drivers_RT_RT600_MIMXRT685S._trng_sample_mode.kTRNG_SampleModeRaw (*C enumerator*), 671
drivers_RT_RT600_MIMXRT685S._trng_sample_mode.kTRNG_SampleModeVonNeumann (*C enumerator*), 671
drivers_RT_RT600_MIMXRT685S._trng_sample_mode.kTRNG_SampleModeVonNeumannRaw (*C enumerator*), 671
drivers_RT_RT600_MIMXRT685S._trng_statistical_check_limit (*C struct*), 673
drivers_RT_RT600_MIMXRT685S._trng_statistical_check_limit.maximum (*C var*), 673
drivers_RT_RT600_MIMXRT685S._trng_statistical_check_limit.minimum (*C var*), 673
drivers_RT_RT600_MIMXRT685S._trng_user_config (*C struct*), 673
drivers_RT_RT600_MIMXRT685S._trng_user_config.clockMode (*C var*), 673
drivers_RT_RT600_MIMXRT685S._trng_user_config.entropyDelay (*C var*), 674
drivers_RT_RT600_MIMXRT685S._trng_user_config.frequencyCountLimit (*C var*), 675
drivers_RT_RT600_MIMXRT685S._trng_user_config.lock (*C var*), 673
drivers_RT_RT600_MIMXRT685S._trng_user_config.longRunMaxLimit (*C var*), 674
drivers_RT_RT600_MIMXRT685S._trng_user_config.monobitLimit (*C var*), 674
drivers_RT_RT600_MIMXRT685S._trng_user_config.pokerLimit (*C var*), 675
drivers_RT_RT600_MIMXRT685S._trng_user_config.retryCount (*C var*), 674
drivers_RT_RT600_MIMXRT685S._trng_user_config.ringOscDiv (*C var*), 673
drivers_RT_RT600_MIMXRT685S._trng_user_config.runBit1Limit (*C var*), 674
drivers_RT_RT600_MIMXRT685S._trng_user_config.runBit2Limit (*C var*), 674
drivers_RT_RT600_MIMXRT685S._trng_user_config.runBit3Limit (*C var*), 674
drivers_RT_RT600_MIMXRT685S._trng_user_config.runBit4Limit (*C var*), 674
drivers_RT_RT600_MIMXRT685S._trng_user_config.runBit5Limit (*C var*), 674
drivers_RT_RT600_MIMXRT685S._trng_user_config.runBit6PlusLimit (*C var*), 674
drivers_RT_RT600_MIMXRT685S._trng_user_config.sampleMode (*C var*), 674
drivers_RT_RT600_MIMXRT685S._trng_user_config.sampleSize (*C var*), 674
drivers_RT_RT600_MIMXRT685S._trng_user_config.sparseBitLimit (*C var*), 674
drivers_RT_RT600_MIMXRT685S._usart_clock_polarity (*C enum*), 688
drivers_RT_RT600_MIMXRT685S._usart_clock_polarity.kUSART_RxSampleOnFallingEdge (*C enumerator*), 688
drivers_RT_RT600_MIMXRT685S._usart_clock_polarity.kUSART_RxSampleOnRisingEdge (*C enumerator*), 688
drivers_RT_RT600_MIMXRT685S._usart_config (*C struct*), 692
drivers_RT_RT600_MIMXRT685S._usart_config.baudRate_Bps (*C var*), 692
drivers_RT_RT600_MIMXRT685S._usart_config.bitCountPerChar (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.clockPolarity (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.enable485ControlOutput (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.enableActiveHighRts (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.enableContinuousSCLK (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.enableHardwareFlowControl (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.enableMode32k (*C var*), 693

drivers_RT_RT600_MIMXRT685S._usart_config.enableRx (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.enableTx (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.extendRtsOutput (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.loopback (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.parityMode (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.rxWatermark (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.stopBitCount (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.syncMode (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_config.txWatermark (*C var*), 693
drivers_RT_RT600_MIMXRT685S._usart_data_len (*C enum*), 688
drivers_RT_RT600_MIMXRT685S._usart_data_len.kUSART_7BitsPerChar (*C enumerator*), 688
drivers_RT_RT600_MIMXRT685S._usart_data_len.kUSART_8BitsPerChar (*C enumerator*), 688
drivers_RT_RT600_MIMXRT685S._usart_dma_handle (*C struct*), 677
drivers_RT_RT600_MIMXRT685S._usart_dma_handle.base (*C var*), 677
drivers_RT_RT600_MIMXRT685S._usart_dma_handle.callback (*C var*), 677
drivers_RT_RT600_MIMXRT685S._usart_dma_handle.rxDataSizeAll (*C var*), 677
drivers_RT_RT600_MIMXRT685S._usart_dma_handle.rxDmaHandle (*C var*), 677
drivers_RT_RT600_MIMXRT685S._usart_dma_handle.rxState (*C var*), 677
drivers_RT_RT600_MIMXRT685S._usart_dma_handle.txDataSizeAll (*C var*), 677
drivers_RT_RT600_MIMXRT685S._usart_dma_handle.txDmaHandle (*C var*), 677
drivers_RT_RT600_MIMXRT685S._usart_dma_handle.txState (*C var*), 677
drivers_RT_RT600_MIMXRT685S._usart_dma_handle.userData (*C var*), 677
drivers_RT_RT600_MIMXRT685S._usart_flags (*C enum*), 690
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_AllClearFlags (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_AutobaudErrorFlag (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_BreakDetectChangeFlag (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_BreakDetectFlag (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_CtsAssertFlag (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_CtsChangeFlag (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_FramingErrorFlag (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_NoiseErrorFlag (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_ParityErrorFlag (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_RxError (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_RxFifoFullFlag (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_RxFifoNotEmptyFlag (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_RxIdleFlag (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_RxStartFlag (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_TxError (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_TxFifoEmptyFlag (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_TxFifoNotFullFlag (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_flags.kUSART_TxIdleFlag (*C enumerator*), 691
drivers_RT_RT600_MIMXRT685S._usart_handle (*C struct*), 694
drivers_RT_RT600_MIMXRT685S._usart_handle.callback (*C var*), 694
drivers_RT_RT600_MIMXRT685S._usart_handle.rxData (*C var*), 694
drivers_RT_RT600_MIMXRT685S._usart_handle.rxDataSize (*C var*), 694
drivers_RT_RT600_MIMXRT685S._usart_handle.rxDataSizeAll (*C var*), 694
drivers_RT_RT600_MIMXRT685S._usart_handle.rxRingBuffer (*C var*), 694
drivers_RT_RT600_MIMXRT685S._usart_handle.rxRingBufferHead (*C var*), 694
drivers_RT_RT600_MIMXRT685S._usart_handle.rxRingBufferSize (*C var*), 694
drivers_RT_RT600_MIMXRT685S._usart_handle.rxRingBufferTail (*C var*), 694
drivers_RT_RT600_MIMXRT685S._usart_handle.rxState (*C var*), 695
drivers_RT_RT600_MIMXRT685S._usart_handle.rxWatermark (*C var*), 695
drivers_RT_RT600_MIMXRT685S._usart_handle.txData (*C var*), 694
drivers_RT_RT600_MIMXRT685S._usart_handle.txDataSize (*C var*), 694
drivers_RT_RT600_MIMXRT685S._usart_handle.txDataSizeAll (*C var*), 694
drivers_RT_RT600_MIMXRT685S._usart_handle.txState (*C var*), 695

drivers_RT_RT600_MIMXRT685S._usart_handle.txWatermark (*C var*), 695
drivers_RT_RT600_MIMXRT685S._usart_handle.userData (*C var*), 695
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable (*C enum*), 689
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_AllInterruptEnables (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_AutoBaudErrorInterruptEnable (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_CtsChangeInterruptEnable (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_FramingErrorInterruptEnable (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_NoiseErrorInterruptEnable (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_ParityErrorInterruptEnable (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_RxBreakChangeInterruptEnable (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_RxErrorInterruptEnable (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_RxLevelInterruptEnable (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_RxStartInterruptEnable (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_TxErrorInterruptEnable (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_TxIdleInterruptEnable (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_interrupt_enable.kUSART_TxLevelInterruptEnable (*C enumerator*), 690
drivers_RT_RT600_MIMXRT685S._usart_parity_mode (*C enum*), 687
drivers_RT_RT600_MIMXRT685S._usart_parity_mode.kUSART_ParityDisabled (*C enumerator*), 687
drivers_RT_RT600_MIMXRT685S._usart_parity_mode.kUSART_ParityEven (*C enumerator*), 687
drivers_RT_RT600_MIMXRT685S._usart_parity_mode.kUSART_ParityOdd (*C enumerator*), 688
drivers_RT_RT600_MIMXRT685S._usart_rxfifo_watermark (*C enum*), 689
drivers_RT_RT600_MIMXRT685S._usart_rxfifo_watermark.kUSART_RxFifo1 (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_rxfifo_watermark.kUSART_RxFifo2 (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_rxfifo_watermark.kUSART_RxFifo3 (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_rxfifo_watermark.kUSART_RxFifo4 (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_rxfifo_watermark.kUSART_RxFifo5 (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_rxfifo_watermark.kUSART_RxFifo6 (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_rxfifo_watermark.kUSART_RxFifo7 (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_rxfifo_watermark.kUSART_RxFifo8 (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_stop_bit_count (*C enum*), 688
drivers_RT_RT600_MIMXRT685S._usart_stop_bit_count.kUSART_OneStopBit (*C enumerator*), 688
drivers_RT_RT600_MIMXRT685S._usart_stop_bit_count.kUSART_TwoStopBit (*C enumerator*), 688
drivers_RT_RT600_MIMXRT685S._usart_sync_mode (*C enum*), 687
drivers_RT_RT600_MIMXRT685S._usart_sync_mode.kUSART_SyncModeDisabled (*C enumera-*

tor), 687
drivers_RT_RT600_MIMXRT685S._usart_sync_mode.kUSART_SyncModeMaster (*C enumerator*), 687
drivers_RT_RT600_MIMXRT685S._usart_sync_mode.kUSART_SyncModeSlave (*C enumerator*), 687
drivers_RT_RT600_MIMXRT685S._usart_transfer (*C struct*), 694
drivers_RT_RT600_MIMXRT685S._usart_transfer.___unnamed19___ (*C union*), 695
drivers_RT_RT600_MIMXRT685S._usart_transfer.___unnamed19___.data (*C var*), 695
drivers_RT_RT600_MIMXRT685S._usart_transfer.___unnamed19___.rxData (*C var*), 695
drivers_RT_RT600_MIMXRT685S._usart_transfer.___unnamed19___.txData (*C var*), 695
drivers_RT_RT600_MIMXRT685S._usart_transfer.dataSize (*C var*), 694
drivers_RT_RT600_MIMXRT685S._usart_txfifo_watermark (*C enum*), 688
drivers_RT_RT600_MIMXRT685S._usart_txfifo_watermark.kUSART_TxFifo0 (*C enumerator*), 688
drivers_RT_RT600_MIMXRT685S._usart_txfifo_watermark.kUSART_TxFifo1 (*C enumerator*), 688
drivers_RT_RT600_MIMXRT685S._usart_txfifo_watermark.kUSART_TxFifo2 (*C enumerator*), 688
drivers_RT_RT600_MIMXRT685S._usart_txfifo_watermark.kUSART_TxFifo3 (*C enumerator*), 688
drivers_RT_RT600_MIMXRT685S._usart_txfifo_watermark.kUSART_TxFifo4 (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_txfifo_watermark.kUSART_TxFifo5 (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_txfifo_watermark.kUSART_TxFifo6 (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usart_txfifo_watermark.kUSART_TxFifo7 (*C enumerator*), 689
drivers_RT_RT600_MIMXRT685S._usdhc_adma2_descriptor (*C struct*), 724
drivers_RT_RT600_MIMXRT685S._usdhc_adma2_descriptor.address (*C var*), 724
drivers_RT_RT600_MIMXRT685S._usdhc_adma2_descriptor.attribute (*C var*), 724
drivers_RT_RT600_MIMXRT685S._usdhc_adma_config (*C struct*), 726
drivers_RT_RT600_MIMXRT685S._usdhc_adma_config.admaTable (*C var*), 726
drivers_RT_RT600_MIMXRT685S._usdhc_adma_config.admaTableWords (*C var*), 726
drivers_RT_RT600_MIMXRT685S._usdhc_adma_config.dmaMode (*C var*), 726
drivers_RT_RT600_MIMXRT685S._usdhc_boot_config (*C struct*), 724
drivers_RT_RT600_MIMXRT685S._usdhc_boot_config.ackTimeoutCount (*C var*), 724
drivers_RT_RT600_MIMXRT685S._usdhc_boot_config.blockCount (*C var*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_boot_config.blockSize (*C var*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_boot_config.bootMode (*C var*), 724
drivers_RT_RT600_MIMXRT685S._usdhc_boot_config.enableAutoStopAtBlockGap (*C var*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_boot_config.enableBootAck (*C var*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_boot_mode (*C enum*), 717
drivers_RT_RT600_MIMXRT685S._usdhc_boot_mode.kUSDHC_BootModeAlternative (*C enumerator*), 717
drivers_RT_RT600_MIMXRT685S._usdhc_boot_mode.kUSDHC_BootModeNormal (*C enumerator*), 717
drivers_RT_RT600_MIMXRT685S._usdhc_burst_len (*C enum*), 720
drivers_RT_RT600_MIMXRT685S._usdhc_burst_len.kUSDHC_EnBurstLenForINCR (*C enumerator*), 720
drivers_RT_RT600_MIMXRT685S._usdhc_burst_len.kUSDHC_EnBurstLenForINCR4816 (*C enumerator*), 720
drivers_RT_RT600_MIMXRT685S._usdhc_burst_len.kUSDHC_EnBurstLenForINCR4816WRAP (*C enumerator*), 720
drivers_RT_RT600_MIMXRT685S._usdhc_capability (*C struct*), 724
drivers_RT_RT600_MIMXRT685S._usdhc_capability.flags (*C var*), 724
drivers_RT_RT600_MIMXRT685S._usdhc_capability.maxBlockCount (*C var*), 724
drivers_RT_RT600_MIMXRT685S._usdhc_capability.maxBlockLength (*C var*), 724
drivers_RT_RT600_MIMXRT685S._usdhc_capability.mmcVersion (*C var*), 724
drivers_RT_RT600_MIMXRT685S._usdhc_capability.sdVersion (*C var*), 724
drivers_RT_RT600_MIMXRT685S._usdhc_card_command_type (*C enum*), 717
drivers_RT_RT600_MIMXRT685S._usdhc_card_command_type.kCARD_CommandTypeAbort (*C enumerator*), 717
drivers_RT_RT600_MIMXRT685S._usdhc_card_command_type.kCARD_CommandTypeEmpty (*C enumerator*), 717

drivers_RT_RT600_MIMXRT685S._usdhc_card_command_type.kCARD_CommandTypeNormal (*C enumerator*), 717
drivers_RT_RT600_MIMXRT685S._usdhc_card_command_type.kCARD_CommandTypeResume (*C enumerator*), 717
drivers_RT_RT600_MIMXRT685S._usdhc_card_command_type.kCARD_CommandTypeSuspend (*C enumerator*), 717
drivers_RT_RT600_MIMXRT685S._usdhc_card_response_type (*C enum*), 717
drivers_RT_RT600_MIMXRT685S._usdhc_card_response_type.kCARD_ResponseTypeNone (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S._usdhc_card_response_type.kCARD_ResponseTypeR1 (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S._usdhc_card_response_type.kCARD_ResponseTypeR1b (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S._usdhc_card_response_type.kCARD_ResponseTypeR2 (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S._usdhc_card_response_type.kCARD_ResponseTypeR3 (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S._usdhc_card_response_type.kCARD_ResponseTypeR4 (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S._usdhc_card_response_type.kCARD_ResponseTypeR5 (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S._usdhc_card_response_type.kCARD_ResponseTypeR5b (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S._usdhc_card_response_type.kCARD_ResponseTypeR6 (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S._usdhc_card_response_type.kCARD_ResponseTypeR7 (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S._usdhc_command (*C struct*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_command.argument (*C var*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_command.flags (*C var*), 726
drivers_RT_RT600_MIMXRT685S._usdhc_command.index (*C var*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_command.response (*C var*), 726
drivers_RT_RT600_MIMXRT685S._usdhc_command.responseErrorFlags (*C var*), 726
drivers_RT_RT600_MIMXRT685S._usdhc_command.responseType (*C var*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_command.type (*C var*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_config (*C struct*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_config.dataTimeout (*C var*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_config.endianMode (*C var*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_config.readWatermarkLevel (*C var*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_config.writeWatermarkLevel (*C var*), 725
drivers_RT_RT600_MIMXRT685S._usdhc_data (*C struct*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_data_bus_width (*C enum*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_data_bus_width.kUSDHC_DataBusWidth1Bit (*C enumerator*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_data_bus_width.kUSDHC_DataBusWidth4Bit (*C enumerator*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_data_bus_width.kUSDHC_DataBusWidth8Bit (*C enumerator*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_data.blockCount (*C var*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_data.blockSize (*C var*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_data.dataType (*C var*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_data.enableAutoCommand12 (*C var*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_data.enableAutoCommand23 (*C var*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_data.enableIgnoreError (*C var*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_data.rxData (*C var*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_data.txData (*C var*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_dma_mode (*C enum*), 716

drivers_RT_RT600_MIMXRT685S._usdhc_dma_mode.kUSDHC_DmaModeAdma1 (*C enumerator*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_dma_mode.kUSDHC_DmaModeAdma2 (*C enumerator*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_dma_mode.kUSDHC_DmaModeSimple (*C enumerator*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_dma_mode.kUSDHC_ExternalDMA (*C enumerator*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_endian_mode (*C enum*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_endian_mode.kUSDHC_EndianModeBig (*C enumerator*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_endian_mode.kUSDHC_EndianModeHalfWordBig (*C enumerator*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_endian_mode.kUSDHC_EndianModeLittle (*C enumerator*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_handle (*C struct*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_handle.callback (*C var*), 729
drivers_RT_RT600_MIMXRT685S._usdhc_handle.command (*C var*), 729
drivers_RT_RT600_MIMXRT685S._usdhc_handle.data (*C var*), 729
drivers_RT_RT600_MIMXRT685S._usdhc_handle.enDMA (*C var*), 729
drivers_RT_RT600_MIMXRT685S._usdhc_handle.transferredWords (*C var*), 729
drivers_RT_RT600_MIMXRT685S._usdhc_handle.userData (*C var*), 729
drivers_RT_RT600_MIMXRT685S._usdhc_host (*C struct*), 729
drivers_RT_RT600_MIMXRT685S._usdhc_host.base (*C var*), 729
drivers_RT_RT600_MIMXRT685S._usdhc_host.capability (*C var*), 729
drivers_RT_RT600_MIMXRT685S._usdhc_host.config (*C var*), 729
drivers_RT_RT600_MIMXRT685S._usdhc_host.sourceClock_Hz (*C var*), 729
drivers_RT_RT600_MIMXRT685S._usdhc_host.transfer (*C var*), 729
drivers_RT_RT600_MIMXRT685S._usdhc_scatter_gather_data (*C struct*), 726
drivers_RT_RT600_MIMXRT685S._usdhc_scatter_gather_data_list (*C struct*), 726
drivers_RT_RT600_MIMXRT685S._usdhc_scatter_gather_data.blockSize (*C var*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_scatter_gather_data.dataDirection (*C var*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_scatter_gather_data.dataType (*C var*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_scatter_gather_data.enableAutoCommand12 (*C var*), 726
drivers_RT_RT600_MIMXRT685S._usdhc_scatter_gather_data.enableAutoCommand23 (*C var*), 726
drivers_RT_RT600_MIMXRT685S._usdhc_scatter_gather_data.enableIgnoreError (*C var*), 726
drivers_RT_RT600_MIMXRT685S._usdhc_scatter_gather_data.sgData (*C var*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_scatter_gather_transfer (*C struct*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_scatter_gather_transfer.command (*C var*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_scatter_gather_transfer.data (*C var*), 727
drivers_RT_RT600_MIMXRT685S._usdhc_transfer (*C struct*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_transfer_callback (*C struct*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_transfer_callback.BlockGap (*C var*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_transfer_callback.CardInserted (*C var*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_transfer_callback.CardRemoved (*C var*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_transfer_callback.ReTuning (*C var*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_transfer_callback.SdioInterrupt (*C var*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_transfer_callback.TransferComplete (*C var*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_transfer_direction (*C enum*), 715
drivers_RT_RT600_MIMXRT685S._usdhc_transfer_direction.kUSDHC_TransferDirectionReceive (*C enumerator*), 715
drivers_RT_RT600_MIMXRT685S._usdhc_transfer_direction.kUSDHC_TransferDirectionSend (*C enumerator*), 716
drivers_RT_RT600_MIMXRT685S._usdhc_transfer.command (*C var*), 728
drivers_RT_RT600_MIMXRT685S._usdhc_transfer.data (*C var*), 728
drivers_RT_RT600_MIMXRT685S._utick_mode (*C enum*), 731

drivers_RT_RT600_MIMXRT685S._utick_mode.kUTICK_Onetime (*C enumerator*), 731
drivers_RT_RT600_MIMXRT685S._utick_mode.kUTICK_Repeat (*C enumerator*), 731
drivers_RT_RT600_MIMXRT685S._wwdt_config (*C struct*), 734
drivers_RT_RT600_MIMXRT685S._wwdt_config.clockFreq_Hz (*C var*), 735
drivers_RT_RT600_MIMXRT685S._wwdt_config.enableWatchdogProtect (*C var*), 734
drivers_RT_RT600_MIMXRT685S._wwdt_config.enableWatchdogReset (*C var*), 734
drivers_RT_RT600_MIMXRT685S._wwdt_config.enableWwdt (*C var*), 734
drivers_RT_RT600_MIMXRT685S._wwdt_config.timeoutValue (*C var*), 734
drivers_RT_RT600_MIMXRT685S._wwdt_config.warningValue (*C var*), 734
drivers_RT_RT600_MIMXRT685S._wwdt_config.windowValue (*C var*), 734
drivers_RT_RT600_MIMXRT685S._wwdt_status_flags_t (*C enum*), 734
drivers_RT_RT600_MIMXRT685S._wwdt_status_flags_t.kWWDT_TimeoutFlag (*C enumerator*), 734
drivers_RT_RT600_MIMXRT685S._wwdt_status_flags_t.kWWDT_WarningFlag (*C enumerator*), 734
drivers_RT_RT600_MIMXRT685S.ACMP0FCLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.ACMP0FCLKSEL_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.acmp_channel_config_t (*C type*), 212
drivers_RT_RT600_MIMXRT685S.ACMP_ClearStatusFlags (*C function*), 209
drivers_RT_RT600_MIMXRT685S.acmp_config_t (*C type*), 212
drivers_RT_RT600_MIMXRT685S.acmp_dac_config_t (*C type*), 212
drivers_RT_RT600_MIMXRT685S.acmp_dac_work_mode_t (*C type*), 212
drivers_RT_RT600_MIMXRT685S.ACMP_Deinit (*C function*), 207
drivers_RT_RT600_MIMXRT685S.ACMP_DisableInterrupts (*C function*), 209
drivers_RT_RT600_MIMXRT685S.acmp_discrete_mode_config_t (*C type*), 212
drivers_RT_RT600_MIMXRT685S.ACMP_Enable (*C function*), 207
drivers_RT_RT600_MIMXRT685S.ACMP_EnableDMA (*C function*), 208
drivers_RT_RT600_MIMXRT685S.ACMP_EnableInterrupts (*C function*), 209
drivers_RT_RT600_MIMXRT685S.ACMP_EnableLinkToDAC (*C function*), 207
drivers_RT_RT600_MIMXRT685S.acmp_filter_config_t (*C type*), 212
drivers_RT_RT600_MIMXRT685S.ACMP_GetDefaultConfig (*C function*), 207
drivers_RT_RT600_MIMXRT685S.ACMP_GetDefaultDiscreteModeConfig (*C function*), 209
drivers_RT_RT600_MIMXRT685S.ACMP_GetStatusFlags (*C function*), 209
drivers_RT_RT600_MIMXRT685S.acmp_hysteresis_mode_t (*C type*), 212
drivers_RT_RT600_MIMXRT685S.ACMP_Init (*C function*), 207
drivers_RT_RT600_MIMXRT685S.acmp_offset_mode_t (*C type*), 211
drivers_RT_RT600_MIMXRT685S.acmp_port_input_t (*C type*), 212
drivers_RT_RT600_MIMXRT685S.acmp_reference_voltage_source_t (*C type*), 212
drivers_RT_RT600_MIMXRT685S.ACMP_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.ACMP_SetChannelConfig (*C function*), 208
drivers_RT_RT600_MIMXRT685S.ACMP_SetDACConfig (*C function*), 208
drivers_RT_RT600_MIMXRT685S.ACMP_SetDiscreteModeConfig (*C function*), 209
drivers_RT_RT600_MIMXRT685S.ACMP_SetFilterConfig (*C function*), 208
drivers_RT_RT600_MIMXRT685S.ADC0FCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.ADC0FCLKSEL0_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.ADC0FCLKSEL1_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.ADC_OFSTRIM_OFSTRIM_MAX (*C macro*), 515
drivers_RT_RT600_MIMXRT685S.ADC_OFSTRIM_OFSTRIM_SIGN (*C macro*), 515
drivers_RT_RT600_MIMXRT685S.ADC_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.AES_DECRYPT (*C macro*), 341
drivers_RT_RT600_MIMXRT685S.AES_ENCRYPT (*C macro*), 341
drivers_RT_RT600_MIMXRT685S.ARRAY_SIZE (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.AT_CACHE_LINE_SECTION (*C macro*), 493
drivers_RT_RT600_MIMXRT685S.AT_CACHE_LINE_SECTION_INIT (*C macro*), 493
drivers_RT_RT600_MIMXRT685S.AT_NONCACHEABLE_SECTION (*C macro*), 491
drivers_RT_RT600_MIMXRT685S.AT_NONCACHEABLE_SECTION_ALIGN (*C macro*), 491
drivers_RT_RT600_MIMXRT685S.AT_NONCACHEABLE_SECTION_ALIGN_INIT (*C macro*), 491

drivers_RT_RT600_MIMXRT685S.AT_NONCACHEABLE_SECTION_INIT (*C macro*), 491
drivers_RT_RT600_MIMXRT685S.AT_QUICKACCESS_SECTION_CODE (*C macro*), 491, 492
drivers_RT_RT600_MIMXRT685S.AT_QUICKACCESS_SECTION_DATA (*C macro*), 492
drivers_RT_RT600_MIMXRT685S.AT_QUICKACCESS_SECTION_DATA_ALIGN (*C macro*), 492
drivers_RT_RT600_MIMXRT685S.audio_pll_mult (*C var*), 263
drivers_RT_RT600_MIMXRT685S.audio_pll_mult_t (*C type*), 250
drivers_RT_RT600_MIMXRT685S.audio_pll_src (*C var*), 263
drivers_RT_RT600_MIMXRT685S.audio_pll_src_t (*C type*), 250
drivers_RT_RT600_MIMXRT685S.AUDIOMCLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.AUDIOMCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.AUDIOPLL0CLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.AUDIOPLLCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.AUX0PLLCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.AUX1PLLCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.base (*C var*), 393, 395
drivers_RT_RT600_MIMXRT685S.body_bias_mode_t (*C type*), 572
drivers_RT_RT600_MIMXRT685S.CACHE64_CleanCache (*C function*), 215
drivers_RT_RT600_MIMXRT685S.CACHE64_CleanCacheByRange (*C function*), 215
drivers_RT_RT600_MIMXRT685S.CACHE64_CleanInvalidateCache (*C function*), 216
drivers_RT_RT600_MIMXRT685S.CACHE64_CleanInvalidateCacheByRange (*C function*), 216
drivers_RT_RT600_MIMXRT685S.CACHE64_CLOCKS (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.cache64_config_t (*C type*), 217
drivers_RT_RT600_MIMXRT685S.CACHE64_DisableCache (*C function*), 215
drivers_RT_RT600_MIMXRT685S.CACHE64_EnableCache (*C function*), 215
drivers_RT_RT600_MIMXRT685S.CACHE64_GetDefaultConfig (*C function*), 214
drivers_RT_RT600_MIMXRT685S.CACHE64_GetInstance (*C function*), 214
drivers_RT_RT600_MIMXRT685S.CACHE64_GetInstanceByAddr (*C function*), 214
drivers_RT_RT600_MIMXRT685S.CACHE64_Init (*C function*), 214
drivers_RT_RT600_MIMXRT685S.CACHE64_InvalidateCache (*C function*), 215
drivers_RT_RT600_MIMXRT685S.CACHE64_InvalidateCacheByRange (*C function*), 215
drivers_RT_RT600_MIMXRT685S.CACHE64_LINESIZE_BYTE (*C macro*), 217
drivers_RT_RT600_MIMXRT685S.cache64_policy_t (*C type*), 217
drivers_RT_RT600_MIMXRT685S.CACHE64_REGION_ALIGNMENT (*C macro*), 218
drivers_RT_RT600_MIMXRT685S.CACHE64_REGION_NUM (*C macro*), 218
drivers_RT_RT600_MIMXRT685S.callback (*C var*), 394, 395
drivers_RT_RT600_MIMXRT685S.casper_algo_t (*C type*), 221
drivers_RT_RT600_MIMXRT685S.CASPER_CLOCKS (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.CASPER_CP (*C macro*), 221
drivers_RT_RT600_MIMXRT685S.CASPER_CP_AREG (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_BREG (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_CREG (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_CTRL0 (*C macro*), 221
drivers_RT_RT600_MIMXRT685S.CASPER_CP_CTRL1 (*C macro*), 221
drivers_RT_RT600_MIMXRT685S.CASPER_CP_DREG (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_ID (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_INTENCLR (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_INTENSET (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_INTSTAT (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_LOADER (*C macro*), 221
drivers_RT_RT600_MIMXRT685S.CASPER_CP_LOCK (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_MASK (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_REMASK (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_RES0 (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_RES1 (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_RES2 (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_RES3 (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_CP_STATUS (*C macro*), 221
drivers_RT_RT600_MIMXRT685S.CASPER_Deinit (*C function*), 221

drivers_RT_RT600_MIMXRT685S.CASPER_ECC_equal (*C function*), 225
drivers_RT_RT600_MIMXRT685S.CASPER_ECC_equal_to_zero (*C function*), 225
drivers_RT_RT600_MIMXRT685S.CASPER_ecc_init (*C function*), 223
drivers_RT_RT600_MIMXRT685S.CASPER_ECC_SECP256R1_Mul (*C function*), 223
drivers_RT_RT600_MIMXRT685S.CASPER_ECC_SECP256R1_MulAdd (*C function*), 223
drivers_RT_RT600_MIMXRT685S.CASPER_ECC_SECP384R1_Mul (*C function*), 224
drivers_RT_RT600_MIMXRT685S.CASPER_ECC_SECP384R1_MulAdd (*C function*), 224
drivers_RT_RT600_MIMXRT685S.CASPER_ECC_SECP521R1_Mul (*C function*), 224
drivers_RT_RT600_MIMXRT685S.CASPER_ECC_SECP521R1_MulAdd (*C function*), 225
drivers_RT_RT600_MIMXRT685S.CASPER_Init (*C function*), 221
drivers_RT_RT600_MIMXRT685S.CASPER_ModExp (*C function*), 222
drivers_RT_RT600_MIMXRT685S.casper_operation_t (*C type*), 221
drivers_RT_RT600_MIMXRT685S.CASPER_Rd32b (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.CASPER_Wr32b (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CASPER_Wr64b (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.CLK_CTL0_PSCCTL0 (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.CLK_CTL0_PSCCTL1 (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.CLK_CTL0_PSCCTL2 (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.CLK_CTL1_PSCCTL0 (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.CLK_CTL1_PSCCTL1 (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.CLK_CTL1_PSCCTL2 (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.CLK_GATE_ABSTRACT_BITS_SHIFT (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.CLK_GATE_ABSTRACT_REG_OFFSET (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.CLK_GATE_BIT_SHIFT_MASK (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.CLK_GATE_BIT_SHIFT_SHIFT (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.CLK_GATE_DEFINE (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.CLK_GATE_REG_OFFSET_MASK (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.CLK_GATE_REG_OFFSET_SHIFT (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.CLKCTL0_TUPLE_MUXA (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.CLKCTL0_TUPLE_MUXB (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.CLKCTL1_TUPLE_MUXA (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.CLKCTL1_TUPLE_MUXB (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.CLKCTL_TUPLE_REG (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.CLKCTL_TUPLE_SEL (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.CLKOUTDIV_OFFSET (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.CLKOUTSEL0_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.CLKOUTSEL1_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.clock_attach_id_t (*C type*), 250
drivers_RT_RT600_MIMXRT685S.CLOCK_AttachClk (*C function*), 251
drivers_RT_RT600_MIMXRT685S.clock_audio_pll_config_t (*C type*), 250
drivers_RT_RT600_MIMXRT685S.CLOCK_DeinitAudioPfd (*C function*), 256
drivers_RT_RT600_MIMXRT685S.CLOCK_DeinitAudioPll (*C function*), 256
drivers_RT_RT600_MIMXRT685S.CLOCK_DeinitSysPfd (*C function*), 255
drivers_RT_RT600_MIMXRT685S.CLOCK_DeinitSysPll (*C function*), 255
drivers_RT_RT600_MIMXRT685S.CLOCK_DisableClock (*C function*), 251
drivers_RT_RT600_MIMXRT685S.clock_div_name_t (*C type*), 250
drivers_RT_RT600_MIMXRT685S.CLOCK_EnableClock (*C function*), 251
drivers_RT_RT600_MIMXRT685S.CLOCK_EnableFfroClk (*C function*), 255
drivers_RT_RT600_MIMXRT685S.CLOCK_EnableOsc32K (*C function*), 254
drivers_RT_RT600_MIMXRT685S.CLOCK_EnableSfroClk (*C function*), 255
drivers_RT_RT600_MIMXRT685S.CLOCK_EnableSysOscClk (*C function*), 253
drivers_RT_RT600_MIMXRT685S.CLOCK_EnableUsbHs0PhyPllClock (*C function*), 256
drivers_RT_RT600_MIMXRT685S.CLOCK_EnableUsbhsDeviceClock (*C function*), 256
drivers_RT_RT600_MIMXRT685S.CLOCK_EnableUsbhsHostClock (*C function*), 256
drivers_RT_RT600_MIMXRT685S.clock_ffro_freq_t (*C type*), 250
drivers_RT_RT600_MIMXRT685S.clock_frg_clk_config_t (*C type*), 251
drivers_RT_RT600_MIMXRT685S.CLOCK_GetAcmpClkFreq (*C function*), 252

drivers_RT_RT600_MIMXRT685S.CLOCK_GetAdcClkFreq (*C function*), 255
drivers_RT_RT600_MIMXRT685S.CLOCK_GetAudioPfdFreq (*C function*), 252
drivers_RT_RT600_MIMXRT685S.CLOCK_GetAudioPllFreq (*C function*), 252
drivers_RT_RT600_MIMXRT685S.CLOCK_GetClockOutClkFreq (*C function*), 255
drivers_RT_RT600_MIMXRT685S.CLOCK_GetCtimerClkFreq (*C function*), 255
drivers_RT_RT600_MIMXRT685S.CLOCK_GetCTimerClkFreq (*C macro*), 256
drivers_RT_RT600_MIMXRT685S.CLOCK_GetDmicClkFreq (*C function*), 253
drivers_RT_RT600_MIMXRT685S.CLOCK_GetDspMainClkFreq (*C function*), 252
drivers_RT_RT600_MIMXRT685S.CLOCK_GetFFroFreq (*C function*), 252
drivers_RT_RT600_MIMXRT685S.CLOCK_GetFlexCommClkFreq (*C function*), 254
drivers_RT_RT600_MIMXRT685S.CLOCK_GetFlexspiClkFreq (*C function*), 255
drivers_RT_RT600_MIMXRT685S.CLOCK_GetFreq (*C function*), 251
drivers_RT_RT600_MIMXRT685S.CLOCK_GetFRGClock (*C function*), 252
drivers_RT_RT600_MIMXRT685S.CLOCK_GetI3cClkFreq (*C function*), 253
drivers_RT_RT600_MIMXRT685S.CLOCK_GetLpOscFreq (*C function*), 254
drivers_RT_RT600_MIMXRT685S.CLOCK_GetMainClkFreq (*C function*), 252
drivers_RT_RT600_MIMXRT685S.CLOCK_GetMclkClkFreq (*C function*), 253
drivers_RT_RT600_MIMXRT685S.CLOCK_GetMclkInClkFreq (*C function*), 254
drivers_RT_RT600_MIMXRT685S.CLOCK_GetOsc32KFreq (*C function*), 254
drivers_RT_RT600_MIMXRT685S.CLOCK_GetSctClkFreq (*C function*), 253
drivers_RT_RT600_MIMXRT685S.CLOCK_GetSdioClkFreq (*C function*), 253
drivers_RT_RT600_MIMXRT685S.CLOCK_GetSFroFreq (*C function*), 252
drivers_RT_RT600_MIMXRT685S.CLOCK_GetSysPfdFreq (*C function*), 252
drivers_RT_RT600_MIMXRT685S.CLOCK_GetSysPllFreq (*C function*), 252
drivers_RT_RT600_MIMXRT685S.CLOCK_GetSystickClkFreq (*C function*), 253
drivers_RT_RT600_MIMXRT685S.CLOCK_GetUsbClkFreq (*C function*), 253
drivers_RT_RT600_MIMXRT685S.CLOCK_GetWakeClk32KFreq (*C function*), 254
drivers_RT_RT600_MIMXRT685S.CLOCK_GetWdtClkFreq (*C function*), 253
drivers_RT_RT600_MIMXRT685S.CLOCK_GetXtalInClkFreq (*C function*), 254
drivers_RT_RT600_MIMXRT685S.CLOCK_InitAudioPfd (*C function*), 256
drivers_RT_RT600_MIMXRT685S.CLOCK_InitAudioPll (*C function*), 255
drivers_RT_RT600_MIMXRT685S.CLOCK_InitSysPfd (*C function*), 255
drivers_RT_RT600_MIMXRT685S.CLOCK_InitSysPll (*C function*), 255
drivers_RT_RT600_MIMXRT685S.clock_ip_name_t (*C type*), 250
drivers_RT_RT600_MIMXRT685S.clock_name_t (*C type*), 250
drivers_RT_RT600_MIMXRT685S.clock_pfd_t (*C type*), 250
drivers_RT_RT600_MIMXRT685S.CLOCK_SetClkDiv (*C function*), 251
drivers_RT_RT600_MIMXRT685S.CLOCK_SetClkinFreq (*C function*), 254
drivers_RT_RT600_MIMXRT685S.CLOCK_SetFRGClock (*C function*), 252
drivers_RT_RT600_MIMXRT685S.CLOCK_SetMclkFreq (*C function*), 254
drivers_RT_RT600_MIMXRT685S.CLOCK_SetXtalFreq (*C function*), 254
drivers_RT_RT600_MIMXRT685S.clock_sys_pll_config_t (*C type*), 250
drivers_RT_RT600_MIMXRT685S.CMP_C0_CFx_MASK (*C macro*), 212
drivers_RT_RT600_MIMXRT685S.CMP_CLOCKS (*C macro*), 256
drivers_RT_RT600_MIMXRT685S.compensation_t (*C type*), 307
drivers_RT_RT600_MIMXRT685S.completionStatus (*C var*), 395
drivers_RT_RT600_MIMXRT685S.CORDIC_ARCTAN (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.CORDIC_ARCTANH (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.CORDIC_ITER (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.CORDIC_MIU (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.CORDIC_T (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.COUNT_TO_MSEC (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.COUNT_TO_USEC (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.CP_CORDIC (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.CP_FFT (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.CP_FIR (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.CP_MTX (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.CP_PQ (*C macro*), 596

drivers_RT_RT600_MIMXRT685S.CRC_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.crc_config_t (*C type*), 264
drivers_RT_RT600_MIMXRT685S.CRC_Deinit (*C function*), 264
drivers_RT_RT600_MIMXRT685S.CRC_DRIVER_USE_CRC16_CCITT_FALSE_AS_DEFAULT (*C macro*), 266
drivers_RT_RT600_MIMXRT685S.CRC_Get16bitResult (*C function*), 265
drivers_RT_RT600_MIMXRT685S.CRC_Get32bitResult (*C function*), 265
drivers_RT_RT600_MIMXRT685S.CRC_GetConfig (*C function*), 265
drivers_RT_RT600_MIMXRT685S.CRC_GetDefaultConfig (*C function*), 265
drivers_RT_RT600_MIMXRT685S.CRC_Init (*C function*), 264
drivers_RT_RT600_MIMXRT685S.crc_polynomial_t (*C type*), 264
drivers_RT_RT600_MIMXRT685S.CRC_Reset (*C function*), 264
drivers_RT_RT600_MIMXRT685S.CRC_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.CRC_WriteData (*C function*), 265
drivers_RT_RT600_MIMXRT685S.CRC_WriteSeed (*C function*), 264
drivers_RT_RT600_MIMXRT685S.CT32BIT0_CAP_PMUX_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.CT32BIT0FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.CT32BIT1_CAP_PMUX_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.CT32BIT1FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.CT32BIT2_CAP_PMUX_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.CT32BIT2FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.CT32BIT3_CAP_PMUX_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.CT32BIT3FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.CT32BIT4_CAP_PMUX_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.CT32BIT4FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.ctimer_callback_t (*C type*), 273
drivers_RT_RT600_MIMXRT685S.ctimer_callback_type_t (*C enum*), 272
drivers_RT_RT600_MIMXRT685S.ctimer_callback_type_t.kCTIMER_MultipleCallback (*C enumerator*), 272
drivers_RT_RT600_MIMXRT685S.ctimer_callback_type_t.kCTIMER_SingleCallback (*C enumerator*), 272
drivers_RT_RT600_MIMXRT685S.ctimer_capture_channel_t (*C type*), 272
drivers_RT_RT600_MIMXRT685S.ctimer_capture_edge_t (*C type*), 272
drivers_RT_RT600_MIMXRT685S.CTIMER_ClearStatusFlags (*C function*), 269
drivers_RT_RT600_MIMXRT685S.CTIMER_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.ctimer_config_t (*C type*), 273
drivers_RT_RT600_MIMXRT685S.CTIMER_Deinit (*C function*), 266
drivers_RT_RT600_MIMXRT685S.CTIMER_DisableInterrupts (*C function*), 269
drivers_RT_RT600_MIMXRT685S.CTIMER_EnableFallingEdgeCapture (*C function*), 275
drivers_RT_RT600_MIMXRT685S.CTIMER_EnableInterrupts (*C function*), 268
drivers_RT_RT600_MIMXRT685S.CTIMER_EnableMatchChannelReload (*C function*), 275
drivers_RT_RT600_MIMXRT685S.CTIMER_EnableResetMatchChannel (*C function*), 275
drivers_RT_RT600_MIMXRT685S.CTIMER_EnableRisingEdgeCapture (*C function*), 275
drivers_RT_RT600_MIMXRT685S.CTIMER_EnableStopMatchChannel (*C function*), 275
drivers_RT_RT600_MIMXRT685S.ctimer_external_match_t (*C type*), 272
drivers_RT_RT600_MIMXRT685S.CTIMER_GetCaptureValue (*C function*), 274
drivers_RT_RT600_MIMXRT685S.CTIMER_GetDefaultConfig (*C function*), 267
drivers_RT_RT600_MIMXRT685S.CTIMER_GetEnabledInterrupts (*C function*), 269
drivers_RT_RT600_MIMXRT685S.CTIMER_GetOutputMatchStatus (*C function*), 273
drivers_RT_RT600_MIMXRT685S.CTIMER_GetStatusFlags (*C function*), 269
drivers_RT_RT600_MIMXRT685S.CTIMER_GetTimerCountValue (*C function*), 274
drivers_RT_RT600_MIMXRT685S.CTIMER_Init (*C function*), 266
drivers_RT_RT600_MIMXRT685S.ctimer_interrupt_enable_t (*C type*), 272
drivers_RT_RT600_MIMXRT685S.ctimer_match_config_t (*C type*), 273
drivers_RT_RT600_MIMXRT685S.ctimer_match_output_control_t (*C type*), 272
drivers_RT_RT600_MIMXRT685S.ctimer_match_t (*C type*), 272
drivers_RT_RT600_MIMXRT685S.CTIMER_RegisterCallBack (*C function*), 274
drivers_RT_RT600_MIMXRT685S.CTIMER_Reset (*C function*), 274

drivers_RT_RT600_MIMXRT685S.CTIMER_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.CTIMER_SetPrescale (*C function*), 274
drivers_RT_RT600_MIMXRT685S.CTIMER_SetShadowValue (*C function*), 276
drivers_RT_RT600_MIMXRT685S.CTIMER_SetupCapture (*C function*), 273
drivers_RT_RT600_MIMXRT685S.CTIMER_SetupMatch (*C function*), 273
drivers_RT_RT600_MIMXRT685S.CTIMER_SetupPwm (*C function*), 267
drivers_RT_RT600_MIMXRT685S.CTIMER_SetupPwmPeriod (*C function*), 267
drivers_RT_RT600_MIMXRT685S.CTIMER_StartTimer (*C function*), 269
drivers_RT_RT600_MIMXRT685S.ctimer_status_flags_t (*C type*), 272
drivers_RT_RT600_MIMXRT685S.CTIMER_StopTimer (*C function*), 269
drivers_RT_RT600_MIMXRT685S.ctimer_timer_mode_t (*C type*), 272
drivers_RT_RT600_MIMXRT685S.CTIMER_UpdatePwmDutycycle (*C function*), 268
drivers_RT_RT600_MIMXRT685S.CTIMER_UpdatePwmPulsePeriod (*C function*), 268
drivers_RT_RT600_MIMXRT685S.dc_removal_t (*C type*), 307
drivers_RT_RT600_MIMXRT685S.DCACHE_CleanByRange (*C function*), 216
drivers_RT_RT600_MIMXRT685S.DCACHE_CleanInvalidateByRange (*C function*), 217
drivers_RT_RT600_MIMXRT685S.DCACHE_InvalidateByRange (*C function*), 216
drivers_RT_RT600_MIMXRT685S.DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.DEBUG_CONSOLE_DEVICE_TYPE_IUART (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.DEBUG_CONSOLE_DEVICE_TYPE_LPSCI (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.DEBUG_CONSOLE_DEVICE_TYPE_LPUART (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.DEBUG_CONSOLE_DEVICE_TYPE_NONE (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.DEBUG_CONSOLE_DEVICE_TYPE_QSCI (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.DEBUG_CONSOLE_DEVICE_TYPE_SWO (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.DEBUG_CONSOLE_DEVICE_TYPE_UART (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.DEBUG_CONSOLE_DEVICE_TYPE_USBCDC (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.DEBUG_CONSOLE_DEVICE_TYPE_VUSART (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.denominator (*C var*), 262, 263
drivers_RT_RT600_MIMXRT685S.destAddr (*C var*), 309
drivers_RT_RT600_MIMXRT685S.DisableDeepSleepIRQ (*C function*), 577
drivers_RT_RT600_MIMXRT685S.DisableGlobalIRQ (*C function*), 503
drivers_RT_RT600_MIMXRT685S.DisableIRQ (*C function*), 503
drivers_RT_RT600_MIMXRT685S.divider (*C var*), 263
drivers_RT_RT600_MIMXRT685S.DMA0_ITRIG_EN0_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.DMA0_ITRIG_PMUX_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.DMA0_OTRIG_PMUX_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.DMA0_REQ_ENA0_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.DMA1_ITRIG_EN0_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.DMA1_ITRIG_PMUX_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.DMA1_OTRIG_PMUX_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.DMA1_REQ_ENA0_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.DMA_AbortTransfer (*C function*), 281
drivers_RT_RT600_MIMXRT685S.DMA_ALLOCATE_DATA_TRANSFER_BUFFER (*C macro*), 292
drivers_RT_RT600_MIMXRT685S.DMA_ALLOCATE_HEAD_DESCRIPTORS (*C macro*), 291
drivers_RT_RT600_MIMXRT685S.DMA_ALLOCATE_HEAD_DESCRIPTORS_AT_NONCACHEABLE (*C macro*), 292
drivers_RT_RT600_MIMXRT685S.DMA_ALLOCATE_LINK_DESCRIPTORS (*C macro*), 292
drivers_RT_RT600_MIMXRT685S.DMA_ALLOCATE_LINK_DESCRIPTORS_AT_NONCACHEABLE (*C macro*), 292
drivers_RT_RT600_MIMXRT685S.dma_burst_wrap_t (*C type*), 291
drivers_RT_RT600_MIMXRT685S.dma_callback (*C type*), 291

drivers_RT_RT600_MIMXRT685S.dma_channel_config_t (*C type*), 291
drivers_RT_RT600_MIMXRT685S.DMA_CHANNEL_GROUP (*C macro*), 292
drivers_RT_RT600_MIMXRT685S.DMA_CHANNEL_INDEX (*C macro*), 292
drivers_RT_RT600_MIMXRT685S.dma_channel_trigger_t (*C type*), 291
drivers_RT_RT600_MIMXRT685S.DMA_CHANNEL_XFER (*C macro*), 293
drivers_RT_RT600_MIMXRT685S.DMA_ChannelIsActive (*C function*), 277
drivers_RT_RT600_MIMXRT685S.DMA_ChannelIsBusy (*C function*), 277
drivers_RT_RT600_MIMXRT685S.DMA_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.DMA_COMMON_CONST_REG_GET (*C macro*), 292
drivers_RT_RT600_MIMXRT685S.DMA_COMMON_REG_GET (*C macro*), 292
drivers_RT_RT600_MIMXRT685S.DMA_COMMON_REG_SET (*C macro*), 292
drivers_RT_RT600_MIMXRT685S.DMA_ConfigureChannelTrigger (*C function*), 278
drivers_RT_RT600_MIMXRT685S.DMA_CreateDescriptor (*C function*), 280
drivers_RT_RT600_MIMXRT685S.DMA_CreateHandle (*C function*), 281
drivers_RT_RT600_MIMXRT685S.DMA_Deinit (*C function*), 277
drivers_RT_RT600_MIMXRT685S.DMA_DESCRIPTOR_END_ADDRESS (*C macro*), 292
drivers_RT_RT600_MIMXRT685S.dma_descriptor_t (*C type*), 290
drivers_RT_RT600_MIMXRT685S.DMA_DisableChannel (*C function*), 278
drivers_RT_RT600_MIMXRT685S.DMA_DisableChannelInterrupts (*C function*), 278
drivers_RT_RT600_MIMXRT685S.DMA_DisableChannelPeriphRq (*C function*), 278
drivers_RT_RT600_MIMXRT685S.DMA_DoChannelSoftwareTrigger (*C function*), 280
drivers_RT_RT600_MIMXRT685S.DMA_DriverIRQHandler (*C function*), 286
drivers_RT_RT600_MIMXRT685S.DMA_EnableChannel (*C function*), 278
drivers_RT_RT600_MIMXRT685S.DMA_EnableChannelInterrupts (*C function*), 277
drivers_RT_RT600_MIMXRT685S.DMA_EnableChannelPeriphRq (*C function*), 278
drivers_RT_RT600_MIMXRT685S.DMA_GetChannelPriority (*C function*), 279
drivers_RT_RT600_MIMXRT685S.DMA_GetRemainingBytes (*C function*), 279
drivers_RT_RT600_MIMXRT685S.dma_handle_t (*C type*), 291
drivers_RT_RT600_MIMXRT685S.DMA_Init (*C function*), 277
drivers_RT_RT600_MIMXRT685S.DMA_InstallDescriptorMemory (*C function*), 277
drivers_RT_RT600_MIMXRT685S.dma_irq_t (*C type*), 291
drivers_RT_RT600_MIMXRT685S.DMA_IRQHandle (*C function*), 286
drivers_RT_RT600_MIMXRT685S.DMA_LoadChannelDescriptor (*C function*), 281
drivers_RT_RT600_MIMXRT685S.DMA_LoadChannelTransferConfig (*C function*), 280
drivers_RT_RT600_MIMXRT685S.DMA_MAX_TRANSFER_COUNT (*C macro*), 291
drivers_RT_RT600_MIMXRT685S.DMA_PrepareChannelTransfer (*C function*), 282
drivers_RT_RT600_MIMXRT685S.DMA_PrepareTransfer (*C function*), 282
drivers_RT_RT600_MIMXRT685S.dma_priority_t (*C type*), 291
drivers_RT_RT600_MIMXRT685S.DMA_RSTS_N (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.DMA_SetCallback (*C function*), 282
drivers_RT_RT600_MIMXRT685S.DMA_SetChannelConfig (*C function*), 279
drivers_RT_RT600_MIMXRT685S.DMA_SetChannelConfigValid (*C function*), 280
drivers_RT_RT600_MIMXRT685S.DMA_SetChannelPriority (*C function*), 279
drivers_RT_RT600_MIMXRT685S.DMA_SetChannelXferConfig (*C function*), 279
drivers_RT_RT600_MIMXRT685S.DMA_SetupChannelDescriptor (*C function*), 281
drivers_RT_RT600_MIMXRT685S.DMA_SetupDescriptor (*C function*), 280
drivers_RT_RT600_MIMXRT685S.DMA_StartTransfer (*C function*), 286
drivers_RT_RT600_MIMXRT685S.DMA_SubmitChannelDescriptor (*C function*), 284
drivers_RT_RT600_MIMXRT685S.DMA_SubmitChannelTransfer (*C function*), 284
drivers_RT_RT600_MIMXRT685S.DMA_SubmitChannelTransferParameter (*C function*), 283
drivers_RT_RT600_MIMXRT685S.DMA_SubmitTransfer (*C function*), 283
drivers_RT_RT600_MIMXRT685S.dma_transfer_config_t (*C type*), 291
drivers_RT_RT600_MIMXRT685S.dma_transfer_type_t (*C type*), 291
drivers_RT_RT600_MIMXRT685S.dma_trigger_burst_t (*C type*), 291
drivers_RT_RT600_MIMXRT685S.dma_trigger_type_t (*C type*), 291
drivers_RT_RT600_MIMXRT685S.dma_xfercfg_t (*C type*), 291
drivers_RT_RT600_MIMXRT685S.DMIC0FCLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.DMIC0FCLKSEL_OFFSET (*C macro*), 260

drivers_RT_RT600_MIMXRT685S.dmic_callback_t (*C type*), 307
drivers_RT_RT600_MIMXRT685S.DMIC_CfgChannelDc (*C function*), 299
drivers_RT_RT600_MIMXRT685S.dmic_channel_config_t (*C type*), 307
drivers_RT_RT600_MIMXRT685S.dmic_channel_t (*C type*), 307
drivers_RT_RT600_MIMXRT685S.DMIC_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.DMIC_ConfigChannel (*C function*), 299
drivers_RT_RT600_MIMXRT685S.DMIC_CtrlClrIntrHwvad (*C function*), 303
drivers_RT_RT600_MIMXRT685S.DMIC_DeInit (*C function*), 298
drivers_RT_RT600_MIMXRT685S.DMIC_DisableChannelGlobalSync (*C function*), 302
drivers_RT_RT600_MIMXRT685S.DMIC_DisableIntCallback (*C function*), 302
drivers_RT_RT600_MIMXRT685S.dmic_dma_handle_t (*C type*), 297
drivers_RT_RT600_MIMXRT685S.dmic_dma_transfer_callback_t (*C type*), 297
drivers_RT_RT600_MIMXRT685S.DMIC_DoFifoReset (*C function*), 301
drivers_RT_RT600_MIMXRT685S.DMIC_EnableChannelDma (*C function*), 300
drivers_RT_RT600_MIMXRT685S.DMIC_EnableChannelFifo (*C function*), 300
drivers_RT_RT600_MIMXRT685S.DMIC_EnableChannelGlobalSync (*C function*), 301
drivers_RT_RT600_MIMXRT685S.DMIC_EnableChannelInterrupt (*C function*), 300
drivers_RT_RT600_MIMXRT685S.DMIC_EnableChannelSignExtend (*C function*), 299
drivers_RT_RT600_MIMXRT685S.DMIC_EnableChannnel (*C function*), 300
drivers_RT_RT600_MIMXRT685S.DMIC_EnableIntCallback (*C function*), 302
drivers_RT_RT600_MIMXRT685S.DMIC_FifoChannel (*C function*), 300
drivers_RT_RT600_MIMXRT685S.DMIC_FifoClearStatus (*C function*), 301
drivers_RT_RT600_MIMXRT685S.DMIC_FifoGetAddress (*C function*), 301
drivers_RT_RT600_MIMXRT685S.DMIC_FifoGetData (*C function*), 301
drivers_RT_RT600_MIMXRT685S.DMIC_FifoGetStatus (*C function*), 301
drivers_RT_RT600_MIMXRT685S.DMIC_FilterResetHwvad (*C function*), 303
drivers_RT_RT600_MIMXRT685S.DMIC_GetInstance (*C function*), 298
drivers_RT_RT600_MIMXRT685S.DMIC_GetNoiseEnvlpEst (*C function*), 303
drivers_RT_RT600_MIMXRT685S.dmic_hwvad_callback_t (*C type*), 307
drivers_RT_RT600_MIMXRT685S.DMIC_HwvadDisableIntCallback (*C function*), 304
drivers_RT_RT600_MIMXRT685S.DMIC_HwvadEnableIntCallback (*C function*), 303
drivers_RT_RT600_MIMXRT685S.DMIC_Init (*C function*), 298
drivers_RT_RT600_MIMXRT685S.DMIC_InstallDMADescriptorMemory (*C function*), 296
drivers_RT_RT600_MIMXRT685S.dmic_phy_sample_rate_t (*C type*), 307
drivers_RT_RT600_MIMXRT685S.DMIC_ResetChannelDecimator (*C function*), 301
drivers_RT_RT600_MIMXRT685S.DMIC_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.DMIC_SetFilterCtrlHwvad (*C function*), 303
drivers_RT_RT600_MIMXRT685S.DMIC_SetGainNoiseEstHwvad (*C function*), 302
drivers_RT_RT600_MIMXRT685S.DMIC_SetGainSignalEstHwvad (*C function*), 302
drivers_RT_RT600_MIMXRT685S.DMIC_SetInputGainHwvad (*C function*), 303
drivers_RT_RT600_MIMXRT685S.DMIC_SetOperationMode (*C function*), 298
drivers_RT_RT600_MIMXRT685S.dmic_transfer_t (*C type*), 297
drivers_RT_RT600_MIMXRT685S.DMIC_TransferAbortReceiveDMA (*C function*), 296
drivers_RT_RT600_MIMXRT685S.DMIC_TransferCreateHandleDMA (*C function*), 295
drivers_RT_RT600_MIMXRT685S.DMIC_TransferGetReceiveCountDMA (*C function*), 296
drivers_RT_RT600_MIMXRT685S.DMIC_TransferReceiveDMA (*C function*), 296
drivers_RT_RT600_MIMXRT685S.DMIC_Use2fs (*C function*), 299
drivers_RT_RT600_MIMXRT685S.dsp_copy_image_t (*C type*), 308
drivers_RT_RT600_MIMXRT685S.DSP_CopyImage (*C function*), 308
drivers_RT_RT600_MIMXRT685S.DSP_Deinit (*C function*), 308
drivers_RT_RT600_MIMXRT685S.DSP_Init (*C function*), 308
drivers_RT_RT600_MIMXRT685S.DSP_INT_PMUX_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.DSP_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.DSP_Start (*C function*), 309
drivers_RT_RT600_MIMXRT685S.DSP_Stop (*C function*), 309
drivers_RT_RT600_MIMXRT685S.DSPCPUCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.DSPCPUCLKSELA_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.DSPCPUCLKSELB_OFFSET (*C macro*), 260

drivers_RT_RT600_MIMXRT685S.DSPMAINRAMCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.DSPPLLCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.ENA_SHIFT (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.EnableDeepSleepIRQ (*C function*), 577
drivers_RT_RT600_MIMXRT685S.EnableGlobalIRQ (*C function*), 503
drivers_RT_RT600_MIMXRT685S.EnableIRQ (*C function*), 503
drivers_RT_RT600_MIMXRT685S.EnableIRQWithPriority (*C function*), 503
drivers_RT_RT600_MIMXRT685S.ESPICLKSEL_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.event (*C var*), 394
drivers_RT_RT600_MIMXRT685S.eventMask (*C var*), 395
drivers_RT_RT600_MIMXRT685S.FC0FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.FC1FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.FC2FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.FC3FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.FC4FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.FC5FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.FC6FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.FC7FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.FC14FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.FC15FCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.flash_run_context_t (*C union*), 454
drivers_RT_RT600_MIMXRT685S.flash_run_context_t.B (*C struct*), 460
drivers_RT_RT600_MIMXRT685S.flash_run_context_t.B (*C var*), 454
drivers_RT_RT600_MIMXRT685S.flash_run_context_t.U (*C var*), 454
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_CLOCKS (*C macro*), 256
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_DriverIRQHandler (*C function*), 310
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_GetInstance (*C function*), 310
drivers_RT_RT600_MIMXRT685S.flexcomm_i2c_dma_master_irq_handler_t (*C type*), 344
drivers_RT_RT600_MIMXRT685S.flexcomm_i2c_master_irq_handler_t (*C type*), 367
drivers_RT_RT600_MIMXRT685S.flexcomm_i2c_slave_irq_handler_t (*C type*), 367
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_Init (*C function*), 310
drivers_RT_RT600_MIMXRT685S.flexcomm_irq_handler_t (*C type*), 310
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_PERIPH_T (*C enum*), 309
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_PERIPH_T.FLEXCOMM_PERIPH_I2C (*C enumerator*), 309
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_PERIPH_T.FLEXCOMM_PERIPH_I2S_RX (*C enumerator*), 310
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_PERIPH_T.FLEXCOMM_PERIPH_I2S_TX (*C enumerator*), 310
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_PERIPH_T.FLEXCOMM_PERIPH_NONE (*C enumerator*), 309
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_PERIPH_T.FLEXCOMM_PERIPH_SPI (*C enumerator*), 309
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_PERIPH_T.FLEXCOMM_PERIPH_USART (*C enumerator*), 309
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.FLEXCOMM_SetIRQHandler (*C function*), 310
drivers_RT_RT600_MIMXRT685S.flexcomm_spi_master_irq_handler_t (*C type*), 665
drivers_RT_RT600_MIMXRT685S.flexcomm_spi_slave_irq_handler_t (*C type*), 665
drivers_RT_RT600_MIMXRT685S.flexcomm_usart_irq_handler_t (*C type*), 692
drivers_RT_RT600_MIMXRT685S.FLEXI2S_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.flexspi_addr_map_config_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.flexspi_ahb_error_code_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.flexspi_ahb_write_wait_unit_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.flexspi_ahbBuffer_config_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.flexspi_ahbBuffers_ctrl_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.flexspi_arb_command_source_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.FLEXSPI_CheckAndClearError (*C function*), 310

drivers_RT_RT600_MIMXRT685S.FLEXSPI_ClearInterruptStatusFlags (*C function*), 313
drivers_RT_RT600_MIMXRT685S.FLEXSPI_CLOCKS (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.flexspi_command_type_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.flexspi_config_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.flexspi_cs_interval_cycle_unit_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.FLEXSPI_Deinit (*C function*), 311
drivers_RT_RT600_MIMXRT685S.flexspi_device_config_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.FLEXSPI_DisableInterrupts (*C function*), 312
drivers_RT_RT600_MIMXRT685S.flexspi_dll_time_t (*C struct*), 454
drivers_RT_RT600_MIMXRT685S.flexspi_dma_callback_t (*C type*), 331
drivers_RT_RT600_MIMXRT685S.flexspi_dma_handle_t (*C type*), 331
drivers_RT_RT600_MIMXRT685S.flexspi_dma_transfer_nsize_t (*C type*), 331
drivers_RT_RT600_MIMXRT685S.FLEXSPI_Enable (*C function*), 311
drivers_RT_RT600_MIMXRT685S.FLEXSPI_EnableInterrupts (*C function*), 312
drivers_RT_RT600_MIMXRT685S.FLEXSPI_EnableRxDMA (*C function*), 312
drivers_RT_RT600_MIMXRT685S.FLEXSPI_EnableTxDMA (*C function*), 312
drivers_RT_RT600_MIMXRT685S.flexspi_flags_t (*C type*), 323
drivers_RT_RT600_MIMXRT685S.FLEXSPI_GetAHBCommandErrorCode (*C function*), 313
drivers_RT_RT600_MIMXRT685S.FLEXSPI_GetArbitratorCommandSource (*C function*), 313
drivers_RT_RT600_MIMXRT685S.FLEXSPI_GetBusIdleStatus (*C function*), 314
drivers_RT_RT600_MIMXRT685S.FLEXSPI_GetDefaultConfig (*C function*), 310
drivers_RT_RT600_MIMXRT685S.FLEXSPI_GetFifoCounts (*C function*), 312
drivers_RT_RT600_MIMXRT685S.FLEXSPI_GetInstance (*C function*), 310
drivers_RT_RT600_MIMXRT685S.FLEXSPI_GetInterruptStatusFlags (*C function*), 313
drivers_RT_RT600_MIMXRT685S.FLEXSPI_GetIPCommandErrorCode (*C function*), 313
drivers_RT_RT600_MIMXRT685S.FLEXSPI_GetRxFifoAddress (*C function*), 312
drivers_RT_RT600_MIMXRT685S.FLEXSPI_GetTxFifoAddress (*C function*), 312
drivers_RT_RT600_MIMXRT685S.flexspi_handle_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.FLEXSPI_Init (*C function*), 310
drivers_RT_RT600_MIMXRT685S.flexspi_ip_error_code_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.FLEXSPI_LUT_SEQ (*C macro*), 324
drivers_RT_RT600_MIMXRT685S.flexspi_lut_seq_t (*C type*), 449
drivers_RT_RT600_MIMXRT685S.flexspi_mem_config_block_t (*C type*), 449
drivers_RT_RT600_MIMXRT685S.flexspi_nor_config_t (*C type*), 450
drivers_RT_RT600_MIMXRT685S.flexspi_operation_t (*C type*), 450
drivers_RT_RT600_MIMXRT685S.flexspi_pad_t (*C type*), 323
drivers_RT_RT600_MIMXRT685S.flexspi_port_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.flexspi_read_sample_clock_t (*C type*), 323
drivers_RT_RT600_MIMXRT685S.FLEXSPI_ReadBlocking (*C function*), 315
drivers_RT_RT600_MIMXRT685S.FLEXSPI_ReadData (*C function*), 314
drivers_RT_RT600_MIMXRT685S.FLEXSPI_ResetFifos (*C function*), 312
drivers_RT_RT600_MIMXRT685S.FLEXSPI_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.FLEXSPI_SetFlashConfig (*C function*), 311
drivers_RT_RT600_MIMXRT685S.FLEXSPI_SoftwareReset (*C function*), 311
drivers_RT_RT600_MIMXRT685S.flexspi_transfer_callback_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.flexspi_transfer_t (*C type*), 324
drivers_RT_RT600_MIMXRT685S.FLEXSPI_TransferAbort (*C function*), 316
drivers_RT_RT600_MIMXRT685S.FLEXSPI_TransferAbortDMA (*C function*), 331
drivers_RT_RT600_MIMXRT685S.FLEXSPI_TransferBlocking (*C function*), 315
drivers_RT_RT600_MIMXRT685S.FLEXSPI_TransferCreateHandle (*C function*), 316
drivers_RT_RT600_MIMXRT685S.FLEXSPI_TransferCreateHandleDMA (*C function*), 330
drivers_RT_RT600_MIMXRT685S.FLEXSPI_TransferDMA (*C function*), 330
drivers_RT_RT600_MIMXRT685S.FLEXSPI_TransferGetCount (*C function*), 316
drivers_RT_RT600_MIMXRT685S.FLEXSPI_TransferGetTransferCountDMA (*C function*), 331
drivers_RT_RT600_MIMXRT685S.FLEXSPI_TransferHandleIRQ (*C function*), 317
drivers_RT_RT600_MIMXRT685S.FLEXSPI_TransferNonBlocking (*C function*), 316
drivers_RT_RT600_MIMXRT685S.FLEXSPI_TransferUpdateSizeDMA (*C function*), 330
drivers_RT_RT600_MIMXRT685S.FLEXSPI_UpdateAhbBuffersSettings (*C function*), 311

drivers_RT_RT600_MIMXRT685S.FLEXSPI_UpdateDllValue (*C function*), 311
drivers_RT_RT600_MIMXRT685S.FLEXSPI_UpdateLUT (*C function*), 314
drivers_RT_RT600_MIMXRT685S.FLEXSPI_UpdateRxSampleClock (*C function*), 314
drivers_RT_RT600_MIMXRT685S.FLEXSPI_WriteBlocking (*C function*), 314
drivers_RT_RT600_MIMXRT685S.FLEXSPI_WriteData (*C function*), 314
drivers_RT_RT600_MIMXRT685S.flexspi_xfer_t (*C type*), 450
drivers_RT_RT600_MIMXRT685S.FLEXSPIFCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.FLEXSPIFCLKSEL_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.FMEAS_GetFrequency (*C function*), 333
drivers_RT_RT600_MIMXRT685S.FMEAS_IsMeasureComplete (*C function*), 332
drivers_RT_RT600_MIMXRT685S.FMEAS_StartMeasure (*C function*), 332
drivers_RT_RT600_MIMXRT685S.FMEAS_SYSCON_FREQMECTRL_CAPVAL (*C macro*), 333
drivers_RT_RT600_MIMXRT685S.FMEAS_SYSCON_FREQMECTRL_CAPVAL_MASK (*C macro*), 333
drivers_RT_RT600_MIMXRT685S.FMEAS_SYSCON_FREQMECTRL_CAPVAL_SHIFT (*C macro*), 333
drivers_RT_RT600_MIMXRT685S.FMEAS_SYSCON_FREQMECTRL_PROG (*C macro*), 333
drivers_RT_RT600_MIMXRT685S.FMEAS_SYSCON_FREQMECTRL_PROG_MASK (*C macro*), 333
drivers_RT_RT600_MIMXRT685S.FMEAS_SYSCON_FREQMECTRL_PROG_SHIFT (*C macro*), 333
drivers_RT_RT600_MIMXRT685S.FMEAS_SYSCON_Type (*C type*), 333
drivers_RT_RT600_MIMXRT685S.FREQME_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.FREQMEAS_PMUX_ID (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.FRGPLLCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.FSL_ACMP_DRIVER_VERSION (*C macro*), 210
drivers_RT_RT600_MIMXRT685S.FSL_CACHE_DRIVER_VERSION (*C macro*), 217
drivers_RT_RT600_MIMXRT685S.FSL_CASPER_DRIVER_VERSION (*C macro*), 218
drivers_RT_RT600_MIMXRT685S.FSL_CLOCK_DRIVER_VERSION (*C macro*), 256
drivers_RT_RT600_MIMXRT685S.FSL_COMMON_DRIVER_VERSION (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.FSL_COMPONENT_ID (*C macro*), 486
drivers_RT_RT600_MIMXRT685S.FSL_CRC_DRIVER_VERSION (*C macro*), 263
drivers_RT_RT600_MIMXRT685S.FSL_CTIMER_DRIVER_VERSION (*C macro*), 269
drivers_RT_RT600_MIMXRT685S.FSL_DMA_DRIVER_VERSION (*C macro*), 286
drivers_RT_RT600_MIMXRT685S.FSL_DMIC_DMA_DRIVER_VERSION (*C macro*), 297
drivers_RT_RT600_MIMXRT685S.FSL_DMIC_DRIVER_VERSION (*C macro*), 304
drivers_RT_RT600_MIMXRT685S.FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ (*C macro*), 504
drivers_RT_RT600_MIMXRT685S.FSL_DSP_DRIVER_VERSION (*C macro*), 309
drivers_RT_RT600_MIMXRT685S.FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE (*C macro*), 291
drivers_RT_RT600_MIMXRT685S.FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT (*C macro*), 324
drivers_RT_RT600_MIMXRT685S.FSL_FLEXCOMM_DRIVER_VERSION (*C macro*), 309
drivers_RT_RT600_MIMXRT685S.FSL_FLEXSPI_DMA_DRIVER_VERSION (*C macro*), 331
drivers_RT_RT600_MIMXRT685S.FSL_FLEXSPI_DRIVER_VERSION (*C macro*), 317
drivers_RT_RT600_MIMXRT685S.FSL_FMEAS_DRIVER_VERSION (*C macro*), 333
drivers_RT_RT600_MIMXRT685S.FSL_GPIO_DRIVER_VERSION (*C macro*), 529
drivers_RT_RT600_MIMXRT685S.FSL_HASHCRYPT_DRIVER_VERSION (*C macro*), 334
drivers_RT_RT600_MIMXRT685S.FSL_I2C_DMA_DRIVER_VERSION (*C macro*), 344
drivers_RT_RT600_MIMXRT685S.FSL_I2C_DRIVER_VERSION (*C macro*), 345
drivers_RT_RT600_MIMXRT685S.FSL_I2S_BRIDGE_DRIVER_VERSION (*C macro*), 372
drivers_RT_RT600_MIMXRT685S.FSL_I2S_DMA_DRIVER_VERSION (*C macro*), 375
drivers_RT_RT600_MIMXRT685S.FSL_I2S_DRIVER_VERSION (*C macro*), 381
drivers_RT_RT600_MIMXRT685S.FSL_I3C_DRIVER_VERSION (*C macro*), 386
drivers_RT_RT600_MIMXRT685S.FSL_IAP_DRIVER_VERSION (*C macro*), 444
drivers_RT_RT600_MIMXRT685S.FSL_INPUTMUX_DRIVER_VERSION (*C macro*), 462
drivers_RT_RT600_MIMXRT685S.FSL_LPADC_DRIVER_VERSION (*C macro*), 524

drivers_RT_RT600_MIMXRT685S.FSL_MRT_DRIVER_VERSION (*C macro*), 534
drivers_RT_RT600_MIMXRT685S.FSL_MU_DRIVER_VERSION (*C macro*), 541
drivers_RT_RT600_MIMXRT685S.FSL_OSTIMER_DRIVER_VERSION (*C macro*), 549
drivers_RT_RT600_MIMXRT685S.FSL_OTFAD_DRIVER_VERSION (*C macro*), 551
drivers_RT_RT600_MIMXRT685S.FSL_PINT_DRIVER_VERSION (*C macro*), 553
drivers_RT_RT600_MIMXRT685S.FSL_POWER_DRIVER_VERSION (*C macro*), 578
drivers_RT_RT600_MIMXRT685S.FSL_POWERQUAD_CMSIS_DRIVER_VERSION (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.FSL_POWERQUAD_DRIVER_VERSION (*C macro*), 594
drivers_RT_RT600_MIMXRT685S.FSL_PUF_DRIVER_VERSION (*C macro*), 612
drivers_RT_RT600_MIMXRT685S.FSL_RESET_DRIVER_VERSION (*C macro*), 619
drivers_RT_RT600_MIMXRT685S.FSL_RTC_DRIVER_VERSION (*C macro*), 625
drivers_RT_RT600_MIMXRT685S.FSL_SCTIMER_DRIVER_VERSION (*C macro*), 640
drivers_RT_RT600_MIMXRT685S.FSL_SEMA42_DRIVER_VERSION (*C macro*), 651
drivers_RT_RT600_MIMXRT685S.FSL_SPI_DMA_DRIVER_VERSION (*C macro*), 657
drivers_RT_RT600_MIMXRT685S.FSL_SPI_DRIVER_VERSION (*C macro*), 659
drivers_RT_RT600_MIMXRT685S.FSL_TRNG_DRIVER_VERSION (*C macro*), 669
drivers_RT_RT600_MIMXRT685S.FSL_USART_DMA_DRIVER_VERSION (*C macro*), 677
drivers_RT_RT600_MIMXRT685S.FSL_USART_DRIVER_VERSION (*C macro*), 686
drivers_RT_RT600_MIMXRT685S.FSL_USDHC_DRIVER_VERSION (*C macro*), 706
drivers_RT_RT600_MIMXRT685S.FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER (*C macro*), 722
drivers_RT_RT600_MIMXRT685S.FSL_UTICK_DRIVER_VERSION (*C macro*), 731
drivers_RT_RT600_MIMXRT685S.FSL_WWDT_DRIVER_VERSION (*C macro*), 733
drivers_RT_RT600_MIMXRT685S.g_clkinFreq (*C var*), 251
drivers_RT_RT600_MIMXRT685S.g_mclkFreq (*C var*), 251
drivers_RT_RT600_MIMXRT685S.g_xtalFreq (*C var*), 251
drivers_RT_RT600_MIMXRT685S.GPIO_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.gpio_interrupt_config_t (*C type*), 530
drivers_RT_RT600_MIMXRT685S.gpio_interrupt_index_t (*C type*), 530
drivers_RT_RT600_MIMXRT685S.gpio_pin_config_t (*C type*), 529
drivers_RT_RT600_MIMXRT685S.gpio_pin_direction_t (*C type*), 529
drivers_RT_RT600_MIMXRT685S.gpio_pin_enable_mode_t (*C type*), 530
drivers_RT_RT600_MIMXRT685S.gpio_pin_enable_polarity_t (*C type*), 530
drivers_RT_RT600_MIMXRT685S.GPIO_PIN_INT_EDGE (*C macro*), 530
drivers_RT_RT600_MIMXRT685S.GPIO_PIN_INT_LEVEL (*C macro*), 530
drivers_RT_RT600_MIMXRT685S.GPIO_PinInit (*C function*), 528
drivers_RT_RT600_MIMXRT685S.GPIO_PinRead (*C function*), 528
drivers_RT_RT600_MIMXRT685S.GPIO_PinWrite (*C function*), 528
drivers_RT_RT600_MIMXRT685S.GPIO_PortClear (*C function*), 530
drivers_RT_RT600_MIMXRT685S.GPIO_PortInit (*C function*), 527
drivers_RT_RT600_MIMXRT685S.GPIO_PortSet (*C function*), 530
drivers_RT_RT600_MIMXRT685S.GPIO_PortToggle (*C function*), 530
drivers_RT_RT600_MIMXRT685S.GPIO_RSTS_N (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_AES_BLOCK_SIZE (*C macro*), 341
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_AES_CryptCtr (*C function*), 339
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_AES_CryptOfb (*C function*), 340
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_AES_DecryptCbc (*C function*), 339
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_AES_DecryptCfb (*C function*), 341
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_AES_DecryptEcb (*C function*), 338
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_AES_EncryptCbc (*C function*), 339
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_AES_EncryptCfb (*C function*), 340
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_AES_EncryptEcb (*C function*), 338
drivers_RT_RT600_MIMXRT685S.hashcrypt_aes_keysize_t (*C type*), 338
drivers_RT_RT600_MIMXRT685S.hashcrypt_aes_mode_t (*C type*), 338
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_AES_SetKey (*C function*), 338
drivers_RT_RT600_MIMXRT685S.hashcrypt_algo_t (*C type*), 336
drivers_RT_RT600_MIMXRT685S.hashcrypt_callback_t (*C type*), 341

drivers_RT_RT600_MIMXRT685S.HASHCRYPT_CLOCKS (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_Deinit (*C function*), 336
drivers_RT_RT600_MIMXRT685S.hashcrypt_handle_t (*C type*), 338
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_HASH_CTX_SIZE (*C macro*), 343
drivers_RT_RT600_MIMXRT685S.hashcrypt_hash_ctx_t (*C type*), 341
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_Init (*C function*), 336
drivers_RT_RT600_MIMXRT685S.hashcrypt_key_t (*C type*), 338
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_MODE_AES (*C macro*), 337
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_MODE_SHA1 (*C macro*), 337
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_MODE_SHA256 (*C macro*), 337
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_SHA (*C function*), 342
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_SHA_Finish (*C function*), 342
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_SHA_Init (*C function*), 342
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_SHA_SetCallback (*C function*), 333
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_SHA_Update (*C function*), 342
drivers_RT_RT600_MIMXRT685S.HASHCRYPT_SHA_UpdateNonBlocking (*C function*), 334
drivers_RT_RT600_MIMXRT685S.I2C_ClearStatusFlags (*C function*), 351
drivers_RT_RT600_MIMXRT685S.I2C_CLOCKS (*C macro*), 256
drivers_RT_RT600_MIMXRT685S.i2c_direction_t (*C type*), 357
drivers_RT_RT600_MIMXRT685S.I2C_DisableInterrupts (*C function*), 352
drivers_RT_RT600_MIMXRT685S.I2C_EnableInterrupts (*C function*), 352
drivers_RT_RT600_MIMXRT685S.I2C_GetEnabledInterrupts (*C function*), 352
drivers_RT_RT600_MIMXRT685S.I2C_GetInstance (*C function*), 350
drivers_RT_RT600_MIMXRT685S.I2C_GetStatusFlags (*C function*), 351
drivers_RT_RT600_MIMXRT685S.i2c_master_config_t (*C type*), 357
drivers_RT_RT600_MIMXRT685S.i2c_master_dma_handle_t (*C type*), 344
drivers_RT_RT600_MIMXRT685S.i2c_master_dma_transfer_callback_t (*C type*), 344
drivers_RT_RT600_MIMXRT685S.i2c_master_handle_t (*C type*), 357
drivers_RT_RT600_MIMXRT685S.i2c_master_transfer_callback_t (*C type*), 357
drivers_RT_RT600_MIMXRT685S.i2c_master_transfer_t (*C type*), 357
drivers_RT_RT600_MIMXRT685S.I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK (*C macro*), 349
drivers_RT_RT600_MIMXRT685S.I2C_MasterClearStatusFlags (*C function*), 351
drivers_RT_RT600_MIMXRT685S.I2C_MasterDeinit (*C function*), 350
drivers_RT_RT600_MIMXRT685S.I2C_MasterEnable (*C function*), 351
drivers_RT_RT600_MIMXRT685S.I2C_MasterGetBusIdleState (*C function*), 353
drivers_RT_RT600_MIMXRT685S.I2C_MasterGetDefaultConfig (*C function*), 350
drivers_RT_RT600_MIMXRT685S.I2C_MasterInit (*C function*), 350
drivers_RT_RT600_MIMXRT685S.I2C_MasterReadBlocking (*C function*), 354
drivers_RT_RT600_MIMXRT685S.I2C_MasterRepeatedStart (*C function*), 353
drivers_RT_RT600_MIMXRT685S.I2C_MasterReset (*C function*), 350
drivers_RT_RT600_MIMXRT685S.I2C_MasterSetBaudRate (*C function*), 352
drivers_RT_RT600_MIMXRT685S.I2C_MasterSetTimeoutValue (*C function*), 352
drivers_RT_RT600_MIMXRT685S.I2C_MasterStart (*C function*), 353
drivers_RT_RT600_MIMXRT685S.I2C_MasterStop (*C function*), 353
drivers_RT_RT600_MIMXRT685S.I2C_MasterTransferAbort (*C function*), 355
drivers_RT_RT600_MIMXRT685S.I2C_MasterTransferAbortDMA (*C function*), 344
drivers_RT_RT600_MIMXRT685S.I2C_MasterTransferBlocking (*C function*), 354
drivers_RT_RT600_MIMXRT685S.I2C_MasterTransferCreateHandle (*C function*), 355
drivers_RT_RT600_MIMXRT685S.I2C_MasterTransferCreateHandleDMA (*C function*), 343
drivers_RT_RT600_MIMXRT685S.I2C_MasterTransferDMA (*C function*), 343
drivers_RT_RT600_MIMXRT685S.I2C_MasterTransferGetCount (*C function*), 355
drivers_RT_RT600_MIMXRT685S.I2C_MasterTransferGetCountDMA (*C function*), 344
drivers_RT_RT600_MIMXRT685S.I2C_MasterTransferHandleIRQ (*C function*), 356
drivers_RT_RT600_MIMXRT685S.I2C_MasterTransferNonBlocking (*C function*), 355
drivers_RT_RT600_MIMXRT685S.I2C_MasterWriteBlocking (*C function*), 354
drivers_RT_RT600_MIMXRT685S.I2C_MAX_DMA_TRANSFER_COUNT (*C macro*), 344

drivers_RT_RT600_MIMXRT685S.I2C_RETRY_TIMES (*C macro*), 349
drivers_RT_RT600_MIMXRT685S.i2c_slave_address_qual_mode_t (*C type*), 366
drivers_RT_RT600_MIMXRT685S.i2c_slave_address_register_t (*C type*), 366
drivers_RT_RT600_MIMXRT685S.i2c_slave_address_t (*C type*), 366
drivers_RT_RT600_MIMXRT685S.i2c_slave_bus_speed_t (*C type*), 366
drivers_RT_RT600_MIMXRT685S.i2c_slave_config_t (*C type*), 366
drivers_RT_RT600_MIMXRT685S.i2c_slave_fsm_t (*C type*), 367
drivers_RT_RT600_MIMXRT685S.i2c_slave_handle_t (*C type*), 367
drivers_RT_RT600_MIMXRT685S.i2c_slave_transfer_callback_t (*C type*), 367
drivers_RT_RT600_MIMXRT685S.i2c_slave_transfer_event_t (*C type*), 366
drivers_RT_RT600_MIMXRT685S.i2c_slave_transfer_t (*C type*), 367
drivers_RT_RT600_MIMXRT685S.I2C_SlaveClearStatusFlags (*C function*), 361
drivers_RT_RT600_MIMXRT685S.I2C_SlaveDeinit (*C function*), 360
drivers_RT_RT600_MIMXRT685S.I2C_SlaveEnable (*C function*), 361
drivers_RT_RT600_MIMXRT685S.I2C_SlaveGetDefaultConfig (*C function*), 360
drivers_RT_RT600_MIMXRT685S.I2C_SlaveGetReceivedAddress (*C function*), 363
drivers_RT_RT600_MIMXRT685S.I2C_SlaveInit (*C function*), 360
drivers_RT_RT600_MIMXRT685S.I2C_SlaveReadBlocking (*C function*), 361
drivers_RT_RT600_MIMXRT685S.I2C_SlaveSetAddress (*C function*), 360
drivers_RT_RT600_MIMXRT685S.I2C_SlaveSetReceiveBuffer (*C function*), 363
drivers_RT_RT600_MIMXRT685S.I2C_SlaveSetSendBuffer (*C function*), 362
drivers_RT_RT600_MIMXRT685S.I2C_SlaveTransferAbort (*C function*), 364
drivers_RT_RT600_MIMXRT685S.I2C_SlaveTransferCreateHandle (*C function*), 362
drivers_RT_RT600_MIMXRT685S.I2C_SlaveTransferGetCount (*C function*), 364
drivers_RT_RT600_MIMXRT685S.I2C_SlaveTransferHandleIRQ (*C function*), 364
drivers_RT_RT600_MIMXRT685S.I2C_SlaveTransferNonBlocking (*C function*), 362
drivers_RT_RT600_MIMXRT685S.I2C_SlaveWriteBlocking (*C function*), 361
drivers_RT_RT600_MIMXRT685S.I2C_STAT_MSTCODE_IDLE (*C macro*), 349
drivers_RT_RT600_MIMXRT685S.I2C_STAT_MSTCODE_NACKADR (*C macro*), 349
drivers_RT_RT600_MIMXRT685S.I2C_STAT_MSTCODE_NACKDAT (*C macro*), 349
drivers_RT_RT600_MIMXRT685S.I2C_STAT_MSTCODE_RXREADY (*C macro*), 349
drivers_RT_RT600_MIMXRT685S.I2C_STAT_MSTCODE_TXREADY (*C macro*), 349
drivers_RT_RT600_MIMXRT685S.I2C_STAT_SLVST_ADDR (*C macro*), 349
drivers_RT_RT600_MIMXRT685S.I2C_STAT_SLVST_RX (*C macro*), 349
drivers_RT_RT600_MIMXRT685S.I2C_STAT_SLVST_TX (*C macro*), 349
drivers_RT_RT600_MIMXRT685S.I2S_BRIDGE_SetFlexcommShareSet (*C function*), 372
drivers_RT_RT600_MIMXRT685S.I2S_BRIDGE_SetFlexcommSignalShareSet (*C function*), 372
drivers_RT_RT600_MIMXRT685S.I2S_BRIDGE_SetShareSetSrc (*C function*), 372
drivers_RT_RT600_MIMXRT685S.I2S_BRIDGE_SetShareSignalSrc (*C function*), 372
drivers_RT_RT600_MIMXRT685S.i2s_bridge_signal_t (*C type*), 372
drivers_RT_RT600_MIMXRT685S.i2s_config_t (*C type*), 383
drivers_RT_RT600_MIMXRT685S.I2S_Deinit (*C function*), 378
drivers_RT_RT600_MIMXRT685S.I2S_Disable (*C function*), 380
drivers_RT_RT600_MIMXRT685S.I2S_DisableInterrupts (*C function*), 381
drivers_RT_RT600_MIMXRT685S.I2S_DisableSecondaryChannel (*C function*), 380
drivers_RT_RT600_MIMXRT685S.i2s_dma_handle_t (*C type*), 375
drivers_RT_RT600_MIMXRT685S.i2s_dma_transfer_callback_t (*C type*), 375
drivers_RT_RT600_MIMXRT685S.I2S_DMACallback (*C function*), 374
drivers_RT_RT600_MIMXRT685S.I2S_EmptyTxFifo (*C function*), 381
drivers_RT_RT600_MIMXRT685S.I2S_Enable (*C function*), 380
drivers_RT_RT600_MIMXRT685S.I2S_EnableInterrupts (*C function*), 380
drivers_RT_RT600_MIMXRT685S.I2S_EnableSecondaryChannel (*C function*), 380
drivers_RT_RT600_MIMXRT685S.i2s_flags_t (*C type*), 383
drivers_RT_RT600_MIMXRT685S.I2S_GetEnabledInterrupts (*C function*), 381
drivers_RT_RT600_MIMXRT685S.i2s_handle_t (*C type*), 383
drivers_RT_RT600_MIMXRT685S.i2s_master_slave_t (*C type*), 383
drivers_RT_RT600_MIMXRT685S.i2s_mode_t (*C type*), 383
drivers_RT_RT600_MIMXRT685S.I2S_NUM_BUFFERS (*C macro*), 384

drivers_RT_RT600_MIMXRT685S.I2S_RxGetDefaultConfig (*C function*), 377
drivers_RT_RT600_MIMXRT685S.I2S_RxHandleIRQ (*C function*), 381
drivers_RT_RT600_MIMXRT685S.I2S_RxInit (*C function*), 377
drivers_RT_RT600_MIMXRT685S.I2S_RxTransferAbort (*C function*), 379
drivers_RT_RT600_MIMXRT685S.I2S_RxTransferCreateHandle (*C function*), 379
drivers_RT_RT600_MIMXRT685S.I2S_RxTransferCreateHandleDMA (*C function*), 373
drivers_RT_RT600_MIMXRT685S.I2S_RxTransferNonBlocking (*C function*), 379
drivers_RT_RT600_MIMXRT685S.I2S_RxTransferReceiveDMA (*C function*), 373
drivers_RT_RT600_MIMXRT685S.I2S_SetBitClockRate (*C function*), 378
drivers_RT_RT600_MIMXRT685S.i2s_transfer_callback_t (*C type*), 383
drivers_RT_RT600_MIMXRT685S.i2s_transfer_t (*C type*), 383
drivers_RT_RT600_MIMXRT685S.I2S_TransferAbortDMA (*C function*), 373
drivers_RT_RT600_MIMXRT685S.I2S_TransferGetCount (*C function*), 379
drivers_RT_RT600_MIMXRT685S.I2S_TransferGetErrorCount (*C function*), 380
drivers_RT_RT600_MIMXRT685S.I2S_TransferInstallLoopDMADescriptorMemory (*C function*), 374
drivers_RT_RT600_MIMXRT685S.I2S_TransferReceiveLoopDMA (*C function*), 375
drivers_RT_RT600_MIMXRT685S.I2S_TransferSendLoopDMA (*C function*), 374
drivers_RT_RT600_MIMXRT685S.I2S_TxGetDefaultConfig (*C function*), 377
drivers_RT_RT600_MIMXRT685S.I2S_TxHandleIRQ (*C function*), 381
drivers_RT_RT600_MIMXRT685S.I2S_TxInit (*C function*), 376
drivers_RT_RT600_MIMXRT685S.I2S_TxTransferAbort (*C function*), 379
drivers_RT_RT600_MIMXRT685S.I2S_TxTransferCreateHandle (*C function*), 378
drivers_RT_RT600_MIMXRT685S.I2S_TxTransferCreateHandleDMA (*C function*), 373
drivers_RT_RT600_MIMXRT685S.I2S_TxTransferNonBlocking (*C function*), 378
drivers_RT_RT600_MIMXRT685S.I2S_TxTransferSendDMA (*C function*), 373
drivers_RT_RT600_MIMXRT685S.I3C0FCLKDIV_OFFSET (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.I3C0FCLKSDIV_OFFSET (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.I3C0FCLKSEL_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.I3C0FCLKSTCDIV_OFFSET (*C macro*), 262
drivers_RT_RT600_MIMXRT685S.I3C0FCLKSTCSEL_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.i3c_baudrate_hz_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.i3c_bus_request_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.i3c_bus_type_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.I3C_CheckForBusyBus (*C function*), 396
drivers_RT_RT600_MIMXRT685S.I3C_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.i3c_config_t (*C type*), 389
drivers_RT_RT600_MIMXRT685S.i3c_device_info_t (*C type*), 388
drivers_RT_RT600_MIMXRT685S.i3c_direction_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.I3C_DriverIRQHandler (*C function*), 413
drivers_RT_RT600_MIMXRT685S.I3C_GetDefaultConfig (*C function*), 389
drivers_RT_RT600_MIMXRT685S.I3C_GetIBIAddress (*C function*), 409
drivers_RT_RT600_MIMXRT685S.I3C_GetIBIType (*C function*), 409
drivers_RT_RT600_MIMXRT685S.I3C_GetInstance (*C function*), 389
drivers_RT_RT600_MIMXRT685S.i3c_hdr_mode_t (*C type*), 388
drivers_RT_RT600_MIMXRT685S.I3C_IBI_BUFF_SIZE (*C macro*), 388
drivers_RT_RT600_MIMXRT685S.i3c_ibi_response_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.i3c_ibi_state_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.i3c_ibi_type_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.I3C_Init (*C function*), 390
drivers_RT_RT600_MIMXRT685S.i3c_legacy_i2c_speed_mode_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.i3c_master_baudrate_config_t (*C type*), 422
drivers_RT_RT600_MIMXRT685S.i3c_master_bus_mode_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.i3c_master_config_t (*C type*), 422
drivers_RT_RT600_MIMXRT685S.i3c_master_daa_baudrate_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.i3c_master_dma_callback_t (*C type*), 392
drivers_RT_RT600_MIMXRT685S.i3c_master_dma_handle_t (*C type*), 392
drivers_RT_RT600_MIMXRT685S.i3c_master_enable_t (*C type*), 420
drivers_RT_RT600_MIMXRT685S.i3c_master_handle_t (*C type*), 422

drivers_RT_RT600_MIMXRT685S.i3c_master_hkeep_t (*C type*), 420
drivers_RT_RT600_MIMXRT685S.i3c_master_isr_t (*C type*), 422
drivers_RT_RT600_MIMXRT685S.i3c_master_src_clk_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.i3c_master_state_t (*C type*), 420
drivers_RT_RT600_MIMXRT685S.i3c_master_transfer_callback_t (*C type*), 422
drivers_RT_RT600_MIMXRT685S.i3c_master_transfer_t (*C type*), 422
drivers_RT_RT600_MIMXRT685S.I3C_MasterCalcTiming (*C function*), 405
drivers_RT_RT600_MIMXRT685S.I3C_MasterCheckAndClearError (*C function*), 396
drivers_RT_RT600_MIMXRT685S.I3C_MasterClearDeviceCount (*C function*), 410
drivers_RT_RT600_MIMXRT685S.I3C_MasterClearErrorStatusFlags (*C function*), 398
drivers_RT_RT600_MIMXRT685S.I3C_MasterClearStatusFlags (*C function*), 398
drivers_RT_RT600_MIMXRT685S.I3C_MasterDeinit (*C function*), 396
drivers_RT_RT600_MIMXRT685S.I3C_MasterDisableInterrupts (*C function*), 400
drivers_RT_RT600_MIMXRT685S.I3C_MasterEmitIBIResponse (*C function*), 409
drivers_RT_RT600_MIMXRT685S.I3C_MasterEmitRequest (*C function*), 408
drivers_RT_RT600_MIMXRT685S.I3C_MasterEnable (*C function*), 396
drivers_RT_RT600_MIMXRT685S.I3C_MasterEnableDMA (*C function*), 402
drivers_RT_RT600_MIMXRT685S.I3C_MasterEnableInterrupts (*C function*), 400
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetBusIdleState (*C function*), 405
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetDefaultConfig (*C function*), 395
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetDeviceListAfterDAA (*C function*), 410
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetEnabledInterrupts (*C function*), 400
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetErrorStatusFlags (*C function*), 398
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetFifoCounts (*C function*), 404
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetI2CPresetTiming (*C function*), 405
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetIBIRules (*C function*), 409
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetPendingInterrupts (*C function*), 401
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetRxFifoAddress (*C function*), 402
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetState (*C function*), 399
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetStatusFlags (*C function*), 397
drivers_RT_RT600_MIMXRT685S.I3C_MasterGetTxFifoAddress (*C function*), 402
drivers_RT_RT600_MIMXRT685S.I3C_MasterInit (*C function*), 396
drivers_RT_RT600_MIMXRT685S.I3C_MasterProcessDAA (*C function*), 410
drivers_RT_RT600_MIMXRT685S.I3C_MasterProcessDAASpecifiedBaudrate (*C function*), 409
drivers_RT_RT600_MIMXRT685S.I3C_MasterReceive (*C function*), 408
drivers_RT_RT600_MIMXRT685S.I3C_MasterRegisterIBI (*C function*), 409
drivers_RT_RT600_MIMXRT685S.I3C_MasterRepeatedStart (*C function*), 407
drivers_RT_RT600_MIMXRT685S.I3C_MasterRepeatedStartWithRxSize (*C function*), 406
drivers_RT_RT600_MIMXRT685S.I3C_MasterSend (*C function*), 407
drivers_RT_RT600_MIMXRT685S.I3C_MasterSetBaudRate (*C function*), 404
drivers_RT_RT600_MIMXRT685S.I3C_MasterSetTiming (*C function*), 405
drivers_RT_RT600_MIMXRT685S.I3C_MasterSetWatermarks (*C function*), 403
drivers_RT_RT600_MIMXRT685S.I3C_MasterStart (*C function*), 406
drivers_RT_RT600_MIMXRT685S.I3C_MasterStartWithRxSize (*C function*), 405
drivers_RT_RT600_MIMXRT685S.I3C_MasterStop (*C function*), 408
drivers_RT_RT600_MIMXRT685S.I3C_MasterTransferAbort (*C function*), 412
drivers_RT_RT600_MIMXRT685S.I3C_MasterTransferAbortDMA (*C function*), 393
drivers_RT_RT600_MIMXRT685S.I3C_MasterTransferBlocking (*C function*), 410
drivers_RT_RT600_MIMXRT685S.I3C_MasterTransferCreateHandle (*C function*), 411
drivers_RT_RT600_MIMXRT685S.I3C_MasterTransferCreateHandleDMA (*C function*), 392
drivers_RT_RT600_MIMXRT685S.I3C_MasterTransferDMA (*C function*), 392
drivers_RT_RT600_MIMXRT685S.I3C_MasterTransferDMAHandleIRQ (*C function*), 393
drivers_RT_RT600_MIMXRT685S.I3C_MasterTransferGetCount (*C function*), 412
drivers_RT_RT600_MIMXRT685S.I3C_MasterTransferGetCountDMA (*C function*), 392
drivers_RT_RT600_MIMXRT685S.I3C_MasterTransferHandleIRQ (*C function*), 412
drivers_RT_RT600_MIMXRT685S.I3C_MasterTransferNonBlocking (*C function*), 412
drivers_RT_RT600_MIMXRT685S.I3C_MasterWaitForCtrlDone (*C function*), 396
drivers_RT_RT600_MIMXRT685S.I3C_MAX_DEVCNT (*C macro*), 388

drivers_RT_RT600_MIMXRT685S.i3c_register_ibi_addr_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.I3C_RETRY_TIMES (*C macro*), 388
drivers_RT_RT600_MIMXRT685S.I3C_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.i3c_rx_term_ops_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.i3c_rx_trigger_level_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.i3c_slave_activity_state_t (*C type*), 440
drivers_RT_RT600_MIMXRT685S.i3c_slave_config_t (*C type*), 440
drivers_RT_RT600_MIMXRT685S.i3c_slave_dma_callback_t (*C type*), 428
drivers_RT_RT600_MIMXRT685S.i3c_slave_dma_handle_t (*C type*), 428
drivers_RT_RT600_MIMXRT685S.i3c_slave_dma_transfer_t (*C type*), 428
drivers_RT_RT600_MIMXRT685S.i3c_slave_event_t (*C type*), 440
drivers_RT_RT600_MIMXRT685S.i3c_slave_handle_t (*C type*), 440
drivers_RT_RT600_MIMXRT685S.i3c_slave_isr_t (*C type*), 440
drivers_RT_RT600_MIMXRT685S.i3c_slave_transfer_callback_t (*C type*), 440
drivers_RT_RT600_MIMXRT685S.i3c_slave_transfer_event_t (*C type*), 440
drivers_RT_RT600_MIMXRT685S.i3c_slave_transfer_t (*C type*), 440
drivers_RT_RT600_MIMXRT685S.I3C_SlaveCheckAndClearError (*C function*), 400
drivers_RT_RT600_MIMXRT685S.I3C_SlaveClearErrorStatusFlags (*C function*), 400, 431
drivers_RT_RT600_MIMXRT685S.I3C_SlaveClearStatusFlags (*C function*), 399, 430
drivers_RT_RT600_MIMXRT685S.I3C_SlaveDeinit (*C function*), 397, 429
drivers_RT_RT600_MIMXRT685S.I3C_SlaveDisableInterrupts (*C function*), 401, 431
drivers_RT_RT600_MIMXRT685S.I3C_SlaveEnable (*C function*), 397, 429
drivers_RT_RT600_MIMXRT685S.I3C_SlaveEnableDMA (*C function*), 403, 432
drivers_RT_RT600_MIMXRT685S.I3C_SlaveEnableInterrupts (*C function*), 401, 431
drivers_RT_RT600_MIMXRT685S.I3C_SlaveGetActivityState (*C function*), 400, 431
drivers_RT_RT600_MIMXRT685S.I3C_SlaveGetDefaultConfig (*C function*), 396, 429
drivers_RT_RT600_MIMXRT685S.I3C_SlaveGetEnabledInterrupts (*C function*), 402, 432
drivers_RT_RT600_MIMXRT685S.I3C_SlaveGetErrorStatusFlags (*C function*), 399, 430
drivers_RT_RT600_MIMXRT685S.I3C_SlaveGetFifoCounts (*C function*), 404, 433
drivers_RT_RT600_MIMXRT685S.I3C_SlaveGetPendingInterrupts (*C function*), 402, 432
drivers_RT_RT600_MIMXRT685S.I3C_SlaveGetRxFifoAddress (*C function*), 403, 433
drivers_RT_RT600_MIMXRT685S.I3C_SlaveGetStatusFlags (*C function*), 399, 429
drivers_RT_RT600_MIMXRT685S.I3C_SlaveGetTxFifoAddress (*C function*), 403, 432
drivers_RT_RT600_MIMXRT685S.I3C_SlaveInit (*C function*), 397, 429
drivers_RT_RT600_MIMXRT685S.I3C_SlaveReceive (*C function*), 411, 433
drivers_RT_RT600_MIMXRT685S.I3C_SlaveSend (*C function*), 411, 433
drivers_RT_RT600_MIMXRT685S.I3C_SlaveSetWatermarks (*C function*), 404, 433
drivers_RT_RT600_MIMXRT685S.I3C_SlaveTransferAbort (*C function*), 435
drivers_RT_RT600_MIMXRT685S.I3C_SlaveTransferAbortDMA (*C function*), 428
drivers_RT_RT600_MIMXRT685S.I3C_SlaveTransferCreateHandle (*C function*), 434
drivers_RT_RT600_MIMXRT685S.I3C_SlaveTransferCreateHandleDMA (*C function*), 427
drivers_RT_RT600_MIMXRT685S.I3C_SlaveTransferDMA (*C function*), 427
drivers_RT_RT600_MIMXRT685S.I3C_SlaveTransferDMAHandleIRQ (*C function*), 428
drivers_RT_RT600_MIMXRT685S.I3C_SlaveTransferGetCount (*C function*), 435
drivers_RT_RT600_MIMXRT685S.I3C_SlaveTransferHandleIRQ (*C function*), 435
drivers_RT_RT600_MIMXRT685S.I3C_SlaveTransferNonBlocking (*C function*), 434
drivers_RT_RT600_MIMXRT685S.i3c_start_scl_delay_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.i3c_timing_t (*C type*), 422
drivers_RT_RT600_MIMXRT685S.i3c_tx_trigger_level_t (*C type*), 421
drivers_RT_RT600_MIMXRT685S.IAP_BOOT_OPTION_MODE_ISP (*C macro*), 443
drivers_RT_RT600_MIMXRT685S.IAP_BOOT_OPTION_MODE_MASTER (*C macro*), 443
drivers_RT_RT600_MIMXRT685S.iap_boot_option_t (*C type*), 443
drivers_RT_RT600_MIMXRT685S.IAP_BOOT_OPTION_TAG (*C macro*), 443
drivers_RT_RT600_MIMXRT685S.IAP_FlexspiConfigClock (*C function*), 452
drivers_RT_RT600_MIMXRT685S.IAP_FlexspiNorAutoConfig (*C function*), 452
drivers_RT_RT600_MIMXRT685S.IAP_FlexspiNorErase (*C function*), 450
drivers_RT_RT600_MIMXRT685S.IAP_FlexspiNorEraseAll (*C function*), 450
drivers_RT_RT600_MIMXRT685S.IAP_FlexspiNorEraseBlock (*C function*), 451

drivers_RT_RT600_MIMXRT685S.IAP_FlexspiNorEraseSector (*C function*), 451
drivers_RT_RT600_MIMXRT685S.IAP_FlexspiNorGetConfig (*C function*), 451
drivers_RT_RT600_MIMXRT685S.IAP_FlexspiNorInit (*C function*), 450
drivers_RT_RT600_MIMXRT685S.IAP_FlexspiNorPageProgram (*C function*), 450
drivers_RT_RT600_MIMXRT685S.IAP_FlexspiNorRead (*C function*), 451
drivers_RT_RT600_MIMXRT685S.IAP_FlexspiSetClockSource (*C function*), 452
drivers_RT_RT600_MIMXRT685S.IAP_FlexspiUpdateLut (*C function*), 452
drivers_RT_RT600_MIMXRT685S.IAP_FlexspiXfer (*C function*), 452
drivers_RT_RT600_MIMXRT685S.IAP_OtpCrcCalc (*C function*), 462
drivers_RT_RT600_MIMXRT685S.IAP_OtpCrcCheck (*C function*), 462
drivers_RT_RT600_MIMXRT685S.IAP_OtpDeinit (*C function*), 461
drivers_RT_RT600_MIMXRT685S.IAP_OtpFuseProgram (*C function*), 461
drivers_RT_RT600_MIMXRT685S.IAP_OtpFuseRead (*C function*), 461
drivers_RT_RT600_MIMXRT685S.IAP_OtpInit (*C function*), 461
drivers_RT_RT600_MIMXRT685S.IAP_OtpShadowRegisterReload (*C function*), 462
drivers_RT_RT600_MIMXRT685S.IAP_RunBootLoader (*C function*), 443
drivers_RT_RT600_MIMXRT685S.ibiAddress (*C var*), 394
drivers_RT_RT600_MIMXRT685S.ibiBuff (*C var*), 394
drivers_RT_RT600_MIMXRT685S.ibiCallback (*C var*), 393
drivers_RT_RT600_MIMXRT685S.ibiPayloadSize (*C var*), 394
drivers_RT_RT600_MIMXRT685S.ibiType (*C var*), 394
drivers_RT_RT600_MIMXRT685S.ICACHE_InvalidateByRange (*C function*), 216
drivers_RT_RT600_MIMXRT685S.INPUTMUX_AttachSignal (*C function*), 483
drivers_RT_RT600_MIMXRT685S.inputmux_connection_t (*C type*), 483
drivers_RT_RT600_MIMXRT685S.INPUTMUX_Deinit (*C function*), 484
drivers_RT_RT600_MIMXRT685S.INPUTMUX_EnableSignal (*C function*), 484
drivers_RT_RT600_MIMXRT685S.INPUTMUX_Init (*C function*), 483
drivers_RT_RT600_MIMXRT685S.INPUTMUX_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.inputmux_signal_t (*C type*), 483
drivers_RT_RT600_MIMXRT685S.INST_BUSY (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.IOPCTL_ANAMUX_EN (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_FULLDRIVE_EN (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC0 (*C macro*), 486
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC1 (*C macro*), 486
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC2 (*C macro*), 486
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC3 (*C macro*), 486
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC4 (*C macro*), 486
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC5 (*C macro*), 486
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC6 (*C macro*), 486
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC7 (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC8 (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC9 (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC10 (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC11 (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC12 (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC13 (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC14 (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_FUNC15 (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.iopctl_group_t (*C type*), 485
drivers_RT_RT600_MIMXRT685S.IOPCTL_INBUF_EN (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_INV_EN (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.IOPCTL_PSEDRAIN_EN (*C macro*), 488
drivers_RT_RT600_MIMXRT685S.IOPCTL_PULLDOWN_EN (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_PULLUP_EN (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_PUPD_EN (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IOPCTL_SLEW_RATE (*C macro*), 487
drivers_RT_RT600_MIMXRT685S.IRQ_ClearPendingIRQ (*C function*), 504
drivers_RT_RT600_MIMXRT685S.IRQ_SetPriority (*C function*), 504

drivers_RT_RT600_MIMXRT685S.isBusy (*C var*), 395
drivers_RT_RT600_MIMXRT685S.KEYSTORE_PUF_DISCHARGE_TIME_FIRST_TRY_MS (*C macro*), 615
drivers_RT_RT600_MIMXRT685S.KEYSTORE_PUF_DISCHARGE_TIME_MAX_MS (*C macro*), 615
drivers_RT_RT600_MIMXRT685S.kFlexcommIrqs (*C var*), 310
drivers_RT_RT600_MIMXRT685S.LPADC_CALIBRATION_READY_TIMEOUT (*C macro*), 515
drivers_RT_RT600_MIMXRT685S.lpadc_calibration_value_t (*C type*), 515
drivers_RT_RT600_MIMXRT685S.LPADC_ClearStatusFlags (*C function*), 517
drivers_RT_RT600_MIMXRT685S.LPADC_ClearTriggerStatusFlags (*C function*), 517
drivers_RT_RT600_MIMXRT685S.LPADC_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.lpadc_config_t (*C struct*), 524
drivers_RT_RT600_MIMXRT685S.lpadc_config_t.conversionAverageMode (*C var*), 524
drivers_RT_RT600_MIMXRT685S.lpadc_config_t.convPauseDelay (*C var*), 525
drivers_RT_RT600_MIMXRT685S.lpadc_config_t.enableAnalogPreliminary (*C var*), 525
drivers_RT_RT600_MIMXRT685S.lpadc_config_t.enableConvPause (*C var*), 525
drivers_RT_RT600_MIMXRT685S.lpadc_config_t.enableInDozeMode (*C var*), 524
drivers_RT_RT600_MIMXRT685S.lpadc_config_t.enableInternalClock (*C var*), 524
drivers_RT_RT600_MIMXRT685S.lpadc_config_t.enableVref1LowVoltage (*C var*), 524
drivers_RT_RT600_MIMXRT685S.lpadc_config_t.FIFOWatermark (*C var*), 525
drivers_RT_RT600_MIMXRT685S.lpadc_config_t.powerLevelMode (*C var*), 525
drivers_RT_RT600_MIMXRT685S.lpadc_config_t.powerUpDelay (*C var*), 525
drivers_RT_RT600_MIMXRT685S.lpadc_config_t.referenceVoltageSource (*C var*), 525
drivers_RT_RT600_MIMXRT685S.lpadc_config_t.triggerPriorityPolicy (*C var*), 525
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t (*C struct*), 525
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.chainedNextCommandNumber (*C var*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.channelBNumber (*C var*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.channelBScaleMode (*C var*), 525
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.channelNumber (*C var*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.conversionResolutionMode (*C var*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.enableAutoChannelIncrement (*C var*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.enableWaitTrigger (*C var*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.hardwareAverageMode (*C var*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.hardwareCompareMode (*C var*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.hardwareCompareValueHigh (*C var*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.hardwareCompareValueLow (*C var*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.loopCount (*C var*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.sampleChannelMode (*C var*), 525
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.sampleScaleMode (*C var*), 525
drivers_RT_RT600_MIMXRT685S.lpadc_conv_command_config_t.sampleTimeMode (*C var*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_result_t (*C struct*), 527
drivers_RT_RT600_MIMXRT685S.lpadc_conv_result_t.commandIdSource (*C var*), 527
drivers_RT_RT600_MIMXRT685S.lpadc_conv_result_t.convValue (*C var*), 527
drivers_RT_RT600_MIMXRT685S.lpadc_conv_result_t.loopCountIndex (*C var*), 527
drivers_RT_RT600_MIMXRT685S.lpadc_conv_result_t.triggerIdSource (*C var*), 527
drivers_RT_RT600_MIMXRT685S.lpadc_conv_trigger_config_t (*C struct*), 526
drivers_RT_RT600_MIMXRT685S.lpadc_conv_trigger_config_t.delayPower (*C var*), 527
drivers_RT_RT600_MIMXRT685S.lpadc_conv_trigger_config_t.enableHardwareTrigger (*C var*), 527
drivers_RT_RT600_MIMXRT685S.lpadc_conv_trigger_config_t.priority (*C var*), 527
drivers_RT_RT600_MIMXRT685S.lpadc_conv_trigger_config_t.targetCommandId (*C var*), 527

drivers_RT_RT600_MIMXRT685S.lpadc_conversion_average_mode_t (*C type*), 514
drivers_RT_RT600_MIMXRT685S.LPADC_CONVERSION_COMPLETE_TIMEOUT (*C macro*), 515
drivers_RT_RT600_MIMXRT685S.lpadc_conversion_resolution_mode_t (*C type*), 514
drivers_RT_RT600_MIMXRT685S.LPADC_Deinit (*C function*), 516
drivers_RT_RT600_MIMXRT685S.LPADC_DisableInterrupts (*C function*), 517
drivers_RT_RT600_MIMXRT685S.LPADC_DoAutoCalibration (*C function*), 520
drivers_RT_RT600_MIMXRT685S.LPADC_DoOffsetCalibration (*C function*), 521
drivers_RT_RT600_MIMXRT685S.LPADC_DoResetConfig (*C function*), 516
drivers_RT_RT600_MIMXRT685S.LPADC_DoResetFIFO (*C function*), 516
drivers_RT_RT600_MIMXRT685S.LPADC_DoSoftwareTrigger (*C function*), 519
drivers_RT_RT600_MIMXRT685S.LPADC_Enable (*C function*), 516
drivers_RT_RT600_MIMXRT685S.LPADC_EnableCalibration (*C function*), 520
drivers_RT_RT600_MIMXRT685S.LPADC_EnableExtraCycle (*C function*), 523
drivers_RT_RT600_MIMXRT685S.LPADC_EnableFIFOWatermarkDMA (*C function*), 518
drivers_RT_RT600_MIMXRT685S.LPADC_EnableHardwareTriggerCommandSelection (*C function*), 519
drivers_RT_RT600_MIMXRT685S.LPADC_EnableHighSpeedConversionMode (*C function*), 523
drivers_RT_RT600_MIMXRT685S.LPADC_EnableInterrupts (*C function*), 517
drivers_RT_RT600_MIMXRT685S.LPADC_EnableJustifiedLeft (*C function*), 524
drivers_RT_RT600_MIMXRT685S.LPADC_EnableOffsetCalibration (*C function*), 521
drivers_RT_RT600_MIMXRT685S.LPADC_FinishAutoCalibration (*C function*), 522
drivers_RT_RT600_MIMXRT685S.LPADC_GAIN_CAL_READY_TIMEOUT (*C macro*), 515
drivers_RT_RT600_MIMXRT685S.LPADC_GET_ACTIVE_COMMAND_STATUS (*C macro*), 515
drivers_RT_RT600_MIMXRT685S.LPADC_GET_ACTIVE_TRIGGER_STATUE (*C macro*), 515
drivers_RT_RT600_MIMXRT685S.LPADC_GetCalibrationValue (*C function*), 522
drivers_RT_RT600_MIMXRT685S.LPADC_GetConvResult (*C function*), 518
drivers_RT_RT600_MIMXRT685S.LPADC_GetConvResultBlocking (*C function*), 518
drivers_RT_RT600_MIMXRT685S.LPADC_GetConvResultCount (*C function*), 518
drivers_RT_RT600_MIMXRT685S.LPADC_GetDefaultConfig (*C function*), 516
drivers_RT_RT600_MIMXRT685S.LPADC_GetDefaultConvCommandConfig (*C function*), 519
drivers_RT_RT600_MIMXRT685S.LPADC_GetDefaultConvTriggerConfig (*C function*), 518
drivers_RT_RT600_MIMXRT685S.LPADC_GetHighSpeedTrimValue (*C function*), 523
drivers_RT_RT600_MIMXRT685S.LPADC_GetOffsetValue (*C function*), 521
drivers_RT_RT600_MIMXRT685S.LPADC_GetStatusFlags (*C function*), 516
drivers_RT_RT600_MIMXRT685S.LPADC_GetTriggerStatusFlags (*C function*), 517
drivers_RT_RT600_MIMXRT685S.LPADC_GetTuneValue (*C function*), 524
drivers_RT_RT600_MIMXRT685S.lpadc_hardware_average_mode_t (*C type*), 513
drivers_RT_RT600_MIMXRT685S.lpadc_hardware_compare_mode_t (*C type*), 514
drivers_RT_RT600_MIMXRT685S.LPADC_Init (*C function*), 516
drivers_RT_RT600_MIMXRT685S.lpadc_offset_calibration_mode_t (*C type*), 514
drivers_RT_RT600_MIMXRT685S.lpadc_power_level_mode_t (*C type*), 514
drivers_RT_RT600_MIMXRT685S.LPADC_PrepareAutoCalibration (*C function*), 522
drivers_RT_RT600_MIMXRT685S.lpadc_reference_voltage_source_t (*C type*), 514
drivers_RT_RT600_MIMXRT685S.LPADC_RequestHighSpeedModeTrim (*C function*), 523
drivers_RT_RT600_MIMXRT685S.lpadc_sample_channel_mode_t (*C type*), 513
drivers_RT_RT600_MIMXRT685S.lpadc_sample_scale_mode_t (*C type*), 513
drivers_RT_RT600_MIMXRT685S.lpadc_sample_time_mode_t (*C type*), 514
drivers_RT_RT600_MIMXRT685S.LPADC_SetCalibrationValue (*C function*), 522
drivers_RT_RT600_MIMXRT685S.LPADC_SetConvCommandConfig (*C function*), 519
drivers_RT_RT600_MIMXRT685S.LPADC_SetConvTriggerConfig (*C function*), 518
drivers_RT_RT600_MIMXRT685S.LPADC_SetHighSpeedTrimValue (*C function*), 523
drivers_RT_RT600_MIMXRT685S.LPADC_SetOffsetCalibrationMode (*C function*), 521
drivers_RT_RT600_MIMXRT685S.LPADC_SetOffsetValue (*C function*), 520, 521
drivers_RT_RT600_MIMXRT685S.LPADC_SetTuneValue (*C function*), 523
drivers_RT_RT600_MIMXRT685S.lpadc_trigger_priority_policy_t (*C type*), 514
drivers_RT_RT600_MIMXRT685S.lpadc_tune_value_t (*C type*), 515

drivers_RT_RT600_MIMXRT685S.LPADC_USE_FIXED_POINT_GAIN_CALCULATION (*C macro*), 515
drivers_RT_RT600_MIMXRT685S.LPC_IOPCTL_DRIVER_VERSION (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.MAINCLKSELA_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.MAINCLKSELB_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.MAINPLLCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.MAKE_PD_BITS (*C macro*), 578
drivers_RT_RT600_MIMXRT685S.MAKE_STATUS (*C macro*), 504
drivers_RT_RT600_MIMXRT685S.MAKE_VERSION (*C macro*), 504
drivers_RT_RT600_MIMXRT685S.MAX (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_DEPRECATED (*C macro*), 492
drivers_RT_RT600_MIMXRT685S.MCUX_DEPRECATED_MACRO (*C macro*), 492
drivers_RT_RT600_MIMXRT685S.MCUX_EXPERIMENTAL (*C macro*), 492
drivers_RT_RT600_MIMXRT685S.MCUX_EXPERIMENTAL_MACRO (*C macro*), 492
drivers_RT_RT600_MIMXRT685S.MCUX_MASK_INVERT_8 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_MASK_INVERT_16 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_MASK_INVERT_32 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_RAMFUNC (*C macro*), 493
drivers_RT_RT600_MIMXRT685S.MCUX_REG_BIT_CLEAR8 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_REG_BIT_CLEAR16 (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.MCUX_REG_BIT_CLEAR32 (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.MCUX_REG_BIT_GET8 (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.MCUX_REG_BIT_GET16 (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.MCUX_REG_BIT_GET32 (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.MCUX_REG_BIT_SET8 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_REG_BIT_SET16 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_REG_BIT_SET32 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_REG_MODIFY8 (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.MCUX_REG_MODIFY16 (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.MCUX_REG_MODIFY32 (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.MCUX_REG_READ8 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_REG_READ16 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_REG_READ32 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_REG_WRITE8 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_REG_WRITE16 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MCUX_REG_WRITE32 (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.MIN (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.mrt_chnl_t (*C type*), 535
drivers_RT_RT600_MIMXRT685S.MRT_ClearStatusFlags (*C function*), 532
drivers_RT_RT600_MIMXRT685S.MRT_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.mrt_config_t (*C type*), 535
drivers_RT_RT600_MIMXRT685S.MRT_Deinit (*C function*), 531
drivers_RT_RT600_MIMXRT685S.MRT_DisableInterrupts (*C function*), 532
drivers_RT_RT600_MIMXRT685S.MRT_EnableInterrupts (*C function*), 532
drivers_RT_RT600_MIMXRT685S.MRT_GetCurrentTimerCount (*C function*), 533
drivers_RT_RT600_MIMXRT685S.MRT_GetDefaultConfig (*C function*), 531
drivers_RT_RT600_MIMXRT685S.MRT_GetEnabledInterrupts (*C function*), 532
drivers_RT_RT600_MIMXRT685S.MRT_GetIdleChannel (*C function*), 534
drivers_RT_RT600_MIMXRT685S.MRT_GetStatusFlags (*C function*), 532
drivers_RT_RT600_MIMXRT685S.MRT_Init (*C function*), 531
drivers_RT_RT600_MIMXRT685S.mrt_interrupt_enable_t (*C type*), 535
drivers_RT_RT600_MIMXRT685S.MRT_ReleaseChannel (*C function*), 534
drivers_RT_RT600_MIMXRT685S.MRT_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.MRT_SetupChannelMode (*C function*), 531
drivers_RT_RT600_MIMXRT685S.MRT_StartTimer (*C function*), 533
drivers_RT_RT600_MIMXRT685S.mrt_status_flags_t (*C type*), 535
drivers_RT_RT600_MIMXRT685S.MRT_StopTimer (*C function*), 533
drivers_RT_RT600_MIMXRT685S.mrt_timer_mode_t (*C type*), 535

drivers_RT_RT600_MIMXRT685S.MRT_UpdateTimerPeriod (*C function*), 532
drivers_RT_RT600_MIMXRT685S.MSDK_HAS_DWT_CYCCNT (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.MSDK_INVALID_IRQ_HANDLER (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.MSDK_REG_NONSECURE_ADDR (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.MSDK_REG_SECURE_ADDR (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.MSEC_TO_COUNT (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.MU_BUSY_POLL_COUNT (*C macro*), 544
drivers_RT_RT600_MIMXRT685S.MU_ClearStatusFlags (*C function*), 540
drivers_RT_RT600_MIMXRT685S.MU_CLOCKS (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.MU_CR_NMI_MASK (*C macro*), 544
drivers_RT_RT600_MIMXRT685S.MU_Deinit (*C function*), 536
drivers_RT_RT600_MIMXRT685S.MU_DisableInterrupts (*C function*), 541
drivers_RT_RT600_MIMXRT685S.MU_EnableInterrupts (*C function*), 540
drivers_RT_RT600_MIMXRT685S.MU_GET_CORE_FLAG (*C macro*), 545
drivers_RT_RT600_MIMXRT685S.MU_GET_GI_FLAG (*C macro*), 545
drivers_RT_RT600_MIMXRT685S.MU_GET_RX_FLAG (*C macro*), 545
drivers_RT_RT600_MIMXRT685S.MU_GET_STAT_FLAG (*C macro*), 545
drivers_RT_RT600_MIMXRT685S.MU_GET_TX_FLAG (*C macro*), 545
drivers_RT_RT600_MIMXRT685S.MU_GetFlags (*C function*), 539
drivers_RT_RT600_MIMXRT685S.MU_GetInterruptsPending (*C function*), 539
drivers_RT_RT600_MIMXRT685S.MU_GetRxStatusFlags (*C function*), 539
drivers_RT_RT600_MIMXRT685S.MU_GetStatusFlags (*C function*), 539
drivers_RT_RT600_MIMXRT685S.MU_Init (*C function*), 536
drivers_RT_RT600_MIMXRT685S.MU_MaskHardwareReset (*C function*), 541
drivers_RT_RT600_MIMXRT685S.mu_msg_reg_index_t (*C type*), 544
drivers_RT_RT600_MIMXRT685S.MU_ReceiveMsg (*C function*), 537
drivers_RT_RT600_MIMXRT685S.MU_ReceiveMsgNonBlocking (*C function*), 537
drivers_RT_RT600_MIMXRT685S.MU_ReceiveMsgTimeout (*C function*), 537
drivers_RT_RT600_MIMXRT685S.MU_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.MU_SendMsg (*C function*), 536
drivers_RT_RT600_MIMXRT685S.MU_SendMsgNonBlocking (*C function*), 536
drivers_RT_RT600_MIMXRT685S.MU_SetFlags (*C function*), 538
drivers_RT_RT600_MIMXRT685S.MU_SetFlagsNonBlocking (*C function*), 538
drivers_RT_RT600_MIMXRT685S.MU_TriggerInterrupts (*C function*), 541
drivers_RT_RT600_MIMXRT685S.mult (*C var*), 263
drivers_RT_RT600_MIMXRT685S.N_wordlen_max (*C macro*), 222
drivers_RT_RT600_MIMXRT685S.NOR_CMD_INDEX_CHIPERASE (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_INDEX_DUMMY (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_INDEX_ERASEBLOCK (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_INDEX_ERASESECTOR (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_INDEX_PAGEPROGRAM (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_INDEX_READ (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_INDEX_READSTATUS (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_INDEX_WRITEENABLE (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_LUT_SEQ_IDX_CHIPERASE (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_LUT_SEQ_IDX_ERASEBLOCK (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_LUT_SEQ_IDX_ERASESECTOR (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_LUT_SEQ_IDX_EXIT_NOCMD (*C macro*), 454
drivers_RT_RT600_MIMXRT685S.NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_LUT_SEQ_IDX_READ (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_LUT_SEQ_IDX_READ_SFDP (*C macro*), 454
drivers_RT_RT600_MIMXRT685S.NOR_CMD_LUT_SEQ_IDX_READSTATUS (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_LUT_SEQ_IDX_READSTATUS_XPI (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.NOR_CMD_LUT_SEQ_IDX_RESTORE_NOCMD (*C macro*), 454
drivers_RT_RT600_MIMXRT685S.NOR_CMD_LUT_SEQ_IDX_WRITEENABLE (*C macro*), 453

drivers_RT_RT600_MIMXRT685S.NOR_CMD_LUT_SEQ_IDX_WRITEENABLE_XPI (*C macro*), 453
drivers_RT_RT600_MIMXRT685S.num (*C var*), 263
drivers_RT_RT600_MIMXRT685S.numerator (*C var*), 262, 263
drivers_RT_RT600_MIMXRT685S.NVIC_ClearPendingIRQ (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.NVIC_DisableIRQ (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.NVIC_EnableIRQ (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.NVIC_GetActive (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.NVIC_GetEnableIRQ (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.NVIC_GetPendingIRQ (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.NVIC_GetPriority (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.NVIC_GetPriorityGrouping (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.NVIC_SetPendingIRQ (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.NVIC_SetPriority (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.NVIC_SetPriorityGrouping (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.operation_mode_t (*C type*), 307
drivers_RT_RT600_MIMXRT685S.OSEVENTFCLKSEL_OFFSET (*C macro*), 260
drivers_RT_RT600_MIMXRT685S.ostimer_callback_t (*C type*), 550
drivers_RT_RT600_MIMXRT685S.OSTIMER_ClearStatusFlags (*C function*), 546
drivers_RT_RT600_MIMXRT685S.OSTIMER_CLOCKS (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.OSTIMER_DecimalToGray (*C function*), 545
drivers_RT_RT600_MIMXRT685S.OSTIMER_Deinit (*C function*), 545
drivers_RT_RT600_MIMXRT685S.OSTIMER_DisableMatchInterrupt (*C function*), 548
drivers_RT_RT600_MIMXRT685S.OSTIMER_DriverIRQHandler (*C function*), 549
drivers_RT_RT600_MIMXRT685S.OSTIMER_EnableMatchInterrupt (*C function*), 548
drivers_RT_RT600_MIMXRT685S.OSTIMER_GetCaptureRawValue (*C function*), 549
drivers_RT_RT600_MIMXRT685S.OSTIMER_GetCaptureValue (*C function*), 549
drivers_RT_RT600_MIMXRT685S.OSTIMER_GetCurrentTimerRawValue (*C function*), 548
drivers_RT_RT600_MIMXRT685S.OSTIMER_GetCurrentTimerValue (*C function*), 549
drivers_RT_RT600_MIMXRT685S.OSTIMER_GetMatchRegister (*C function*), 548
drivers_RT_RT600_MIMXRT685S.OSTIMER_GetMatchValue (*C function*), 548
drivers_RT_RT600_MIMXRT685S.OSTIMER_GetStatusFlags (*C function*), 545
drivers_RT_RT600_MIMXRT685S.OSTIMER_GrayToDecimal (*C function*), 545
drivers_RT_RT600_MIMXRT685S.OSTIMER_HandleIRQ (*C function*), 549
drivers_RT_RT600_MIMXRT685S.OSTIMER_Init (*C function*), 545
drivers_RT_RT600_MIMXRT685S.OSTIMER_RSTS (*C macro*), 621
drivers_RT_RT600_MIMXRT685S.OSTIMER_SetMatchRawValue (*C function*), 546
drivers_RT_RT600_MIMXRT685S.OSTIMER_SetMatchRegister (*C function*), 547
drivers_RT_RT600_MIMXRT685S.OSTIMER_SetMatchValue (*C function*), 546
drivers_RT_RT600_MIMXRT685S.OSTIMER_SetMatchValueSafe (*C function*), 546
drivers_RT_RT600_MIMXRT685S.otfad_config_t (*C type*), 552
drivers_RT_RT600_MIMXRT685S.OTFAD_Deinit (*C function*), 550
drivers_RT_RT600_MIMXRT685S.otfad_encryption_config_t (*C type*), 552
drivers_RT_RT600_MIMXRT685S.OTFAD_GetDefaultConfig (*C function*), 550
drivers_RT_RT600_MIMXRT685S.OTFAD_GetEncryptionConfig (*C function*), 551
drivers_RT_RT600_MIMXRT685S.OTFAD_GetOperateMode (*C function*), 550
drivers_RT_RT600_MIMXRT685S.OTFAD_GetStatus (*C function*), 550
drivers_RT_RT600_MIMXRT685S.OTFAD_HitDetermination (*C function*), 551
drivers_RT_RT600_MIMXRT685S.OTFAD_Init (*C function*), 550
drivers_RT_RT600_MIMXRT685S.OTFAD_SetEncryptionConfig (*C function*), 550
drivers_RT_RT600_MIMXRT685S.pd_bit_t (*C type*), 572
drivers_RT_RT600_MIMXRT685S.pd_bits (*C enum*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_CASPER_SRAM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_DSP_CACHE_REGF (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_DSP_TCM_REGF (*C enumerator*), 566

drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_FLEXSPI_SRAM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_PQ_SRAM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF0 (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF1 (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF2 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF3 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF4 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF5 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF6 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF7 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF8 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF9 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF10 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF11 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF12 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF13 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF14 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF15 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF16 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF17 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF18 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF19 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF20 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF21 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF22 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF23 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF24 (*C enumerator*), 567
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF25 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF26 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF27 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF28 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_SRAM_IF29 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_USBHS_SRAM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_USDHC0_SRAM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_APD_USDHC1_SRAM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_ForceUnsigned (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_LP_ADC (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_LP_HSPAD_VDET (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_LP_LVDCORE (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_LP_PMCREF (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_LP_PORCORE (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_LP_VDD_COREREG (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_ACMP (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_ADC (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_ADC_TEMPSNS (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_AUDPLL_ANA (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_AUDPLL_LDO (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_FFRO (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_HSPAD_REF (*C enumerator*), 566

drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_HVD1V8 (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_HVDCORE (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_LPOSC (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_ROM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_SFRO (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_SYSPLL_ANA (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_SYSPLL_LDO (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PD_SYSXTAL (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PMC_MODE0 (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PMC_MODE1 (*C enumerator*), 565
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_CASPER_SRAM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_DSP_CACHE_REGF (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_DSP_TCM_REGF (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_FLEXSPI_SRAM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_PQ_SRAM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF0 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF1 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF2 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF3 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF4 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF5 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF6 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF7 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF8 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF9 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF10 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF11 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF12 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF13 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF14 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF15 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF16 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF17 (*C enumerator*), 568
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF18 (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF19 (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF20 (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF21 (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF22 (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF23 (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF24 (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF25 (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF26 (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF27 (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF28 (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_SRAM_IF29 (*C enumerator*), 569
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_USBHS_SRAM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_USDHC0_SRAM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_PPD_USDHC1_SRAM (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pd_bits.kPDRUNCFG_SRAM_SLEEP (*C enumerator*), 566
drivers_RT_RT600_MIMXRT685S.pdm_div_t (*C enum*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv1 (*C enumerator*), 305

drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv2 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv3 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv4 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv6 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv8 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv12 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv16 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv24 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv32 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv48 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv64 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv96 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.pdm_div_t.kDMIC_PdmDiv128 (*C enumerator*), 305
drivers_RT_RT600_MIMXRT685S.PDRCFG0 (*C macro*), 578
drivers_RT_RT600_MIMXRT685S.PDRCFG1 (*C macro*), 578
drivers_RT_RT600_MIMXRT685S.PDRCFG2 (*C macro*), 578
drivers_RT_RT600_MIMXRT685S.PDRCFG3 (*C macro*), 578
drivers_RT_RT600_MIMXRT685S.PFC0CLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.PFC1CLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kCASPER_RamOffset_Base (*C enumerator*), 221
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kCASPER_RamOffset_M64 (*C enumerator*), 221
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kCASPER_RamOffset_Modulus (*C enumerator*), 221
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kCASPER_RamOffset_Result (*C enumerator*), 221
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kCASPER_RamOffset_TempBase (*C enumerator*), 221
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kCLOCK_FrgFFro (*C enumerator*), 262
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kCLOCK_FrgMainClk (*C enumerator*), 262
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kCLOCK_FrgPllDiv (*C enumerator*), 262
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kCLOCK_FrgSFro (*C enumerator*), 262
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDeviceConfigCmdType_Generic (*C enumerator*), 449
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDeviceConfigCmdType_QuadEnable (*C enumerator*), 449
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDeviceConfigCmdType_Reset (*C enumerator*), 449
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDeviceConfigCmdType_Spi2NoCmd (*C enumerator*), 449
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDeviceConfigCmdType_Spi2Xpi (*C enumerator*), 449
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDeviceConfigCmdType_Xpi2Spi (*C enumerator*), 449
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_AddressInterleave0xWidth (*C enumerator*), 286
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_AddressInterleave1xWidth (*C enumerator*), 286
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_AddressInterleave2xWidth (*C enumerator*), 286
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_AddressInterleave4xWidth (*C enumerator*), 287
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_BurstSize1 (*C enumerator*), 288
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_BurstSize2 (*C enumerator*), 288
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_BurstSize4 (*C enumerator*), 288

drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_BurstSize8 (*C enumerator*), 288
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_BurstSize16 (*C enumerator*), 288
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_BurstSize32 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_BurstSize64 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_BurstSize128 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_BurstSize256 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_BurstSize512 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_BurstSize1024 (*C enumerator*), 289
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_Transfer8BitWidth (*C enumerator*), 287
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_Transfer16BitWidth (*C enumerator*), 287
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMA_Transfer32BitWidth (*C enumerator*), 287
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMIC_EnableChannel0 (*C enumerator*), 306
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kDMIC_EnableChannel1 (*C enumerator*), 307
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlashInstMode_0_4_4_DDR (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlashInstMode_0_4_4_SDR (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlashInstMode_ExtendedSpi (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlashInstMode_OPI_DDR (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlashInstMode_OPI_SDR (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlashInstMode_QPI_DDR (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlashInstMode_QPI_SDR (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlashResetLogic_Disabled (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlashResetLogic_JedecHwReset (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlashResetLogic_ResetPin (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_CADDR_DDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_CADDR_SDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_DATSZ_DDR (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_DATSZ_SDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_DDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_DUMMY_DDR (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_DUMMY_RWDS_DDR (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_DUMMY_RWDS_SDR (*C enumerator*), 318

drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_DUMMY_SDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_JUMP_ON_CS (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_LEARN_DDR (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_LEARN_SDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_MODE1_DDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_MODE1_SDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_MODE2_DDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_MODE2_SDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_MODE4_DDR (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_MODE4_SDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_MODE8_DDR (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_MODE8_SDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_RADDR_DDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_RADDR_SDR (*C enumerator*), 317
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_READ_DDR (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_READ_SDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_SDR (*C enumerator*), 317
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_STOP (*C enumerator*), 317
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_WRITE_DDR (*C enumerator*), 319
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFLEXSPI_Command_WRITE_SDR (*C enumerator*), 318
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlexSpiDeviceType_MCP_NOR_NAND (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlexSpiDeviceType_MCP_NOR_RAM (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlexSpiDeviceType_SerialNAND (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlexSpiDeviceType_SerialNOR (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kFlexSpiDeviceType_SerialRAM (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kI2S_SecondaryChannel1 (*C enumerator*), 383
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kI2S_SecondaryChannel2 (*C enumerator*), 383
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kI2S_SecondaryChannel3 (*C enumerator*), 383
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kOTFAD_Context_0 (*C enumerator*), 551

drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kOTFAD_Context_1 (*C enumerator*), 551
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kOTFAD_Context_2 (*C enumerator*), 552
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kOTFAD_Context_3 (*C enumerator*), 552
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kOTFAD_LDM (*C enumerator*), 552
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kOTFAD_NRM (*C enumerator*), 552
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kOTFAD_SVM (*C enumerator*), 552
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kRestoreSequence_4QPI_FF (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kRestoreSequence_5QPI_FF (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kRestoreSequence_8QPI_FF (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kRestoreSequence_HW_Reset (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kRestoreSequence_None (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kRestoreSequence_Send_06_FF (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kRestoreSequence_Send_66_99 (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kRestoreSequence_Send_6699_9966 (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kRestoreSequence_Send_F0 (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialFlash_1Pad (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialFlash_2Pads (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialFlash_4Pads (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialFlash_8Pads (*C enumerator*), 448
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorCfgOption_DeviceType_AdestoOctalDDR (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorCfgOption_DeviceType_AdestoOctalSDR (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorCfgOption_DeviceType_HyperFLASH1V8 (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorCfgOption_DeviceType_HyperFLASH3V0 (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorCfgOption_DeviceType_MacronixOctalDDR (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorCfgOption_DeviceType_MacronixOctalSDR (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorCfgOption_DeviceType_MicronOctalDDR (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorCfgOption_DeviceType_MicronOctalSDR (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorCfgOption_DeviceType_ReadSFDP_DDR (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorCfgOption_DeviceType_ReadSFDP_SDR (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorCfgOption_Tag (*C enumerator*), 445
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorConnection_BothPorts (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorConnection_Parallel (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorConnection_SinglePortA (*C*

enumerator), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorConnection_SinglePortB (*C enumerator*), 447
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorEnhanceMode_0_4_4_Mode (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorEnhanceMode_0_8_8_Mode (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorEnhanceMode_2ndPinMux (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorEnhanceMode_DataOrderSwapped (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorEnhanceMode_Disabled (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorQuadMode_NotConfig (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorQuadMode_StatusReg1_Bit6 (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorQuadMode_StatusReg2_Bit1 (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorQuadMode_StatusReg2_Bit1_0x31 (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kSerialNorQuadMode_StatusReg2_Bit7 (*C enumerator*), 446
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_Busy (*C enumerator*), 502
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_DMA_Busy (*C enumerator*), 286
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_DMIC_Busy (*C enumerator*), 304
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_DMIC_Idle (*C enumerator*), 304
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_DMIC_OverRunError (*C enumerator*), 304
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_DMIC_UnderRunError (*C enumerator*), 304
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_EnrollNotAllowed (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_Fail (*C enumerator*), 502
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_FLEXSPI_Busy (*C enumerator*), 317
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_FLEXSPI_IpCommandGrantTimeout (*C enumerator*), 317
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_FLEXSPI_IpCommandSequenceError (*C enumerator*), 317
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_FLEXSPI_SequenceExecutionTimeout (*C enumerator*), 317
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_Addr_Nak (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_ArbitrationLost (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_BitError (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_Busy (*C enumerator*), 345
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_DmaRequestFail (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_EventTimeout (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_Idle (*C enumerator*), 345
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_InvalidParameter (*C enu-*

merator), 345
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_Nak (*C enumerator*), 345
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_NoTransferInProgress (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_SclLowTimeout (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_StartStopError (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_Timeout (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2C_UnexpectedState (*C enumerator*), 346
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2S_BufferComplete (*C enumerator*), 381
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2S_Busy (*C enumerator*), 382
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I2S_Done (*C enumerator*), 382
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_Busy (*C enumerator*), 386
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_CrcError (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_HdrParityError (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_IBIWon (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_Idle (*C enumerator*), 386
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_InvalidReq (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_InvalidStart (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_MsgError (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_Nak (*C enumerator*), 386
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_OverrunError (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_ReadFifoError (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_S0S1Error (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_SdrParityError (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_SlaveCountExceed (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_Term (*C enumerator*), 386
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_Timeout (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_UnderrunError (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_UnderrunNak (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_WriteAbort (*C enumerator*), 386
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_I3C_WriteFifoError (*C enumerator*), 387
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_InvalidArgument (*C enumerator*), 502
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_NoData (*C enumerator*), 502
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_NoTransferInProgress (*C enumerator*), 502
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OTFAD_AddressError (*C enu-*

merator), 551
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OTFAD_RegionMiss (*C enumerator*), 551
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OTFAD_RegionOverlap (*C enumerator*), 551
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OTFAD_ResRegAccessMode (*C enumerator*), 551
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OTP_CrcCheckPass (*C enumerator*), 461
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OTP_CrcFail (*C enumerator*), 461
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OTP_EccCheckFail (*C enumerator*), 461
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OTP_Error (*C enumerator*), 461
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OTP_InvalidAddress (*C enumerator*), 460
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OTP_Locked (*C enumerator*), 461
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OTP_ProgramFail (*C enumerator*), 460
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OTP_Timeout (*C enumerator*), 461
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_OutOfRange (*C enumerator*), 502
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_ReadOnly (*C enumerator*), 502
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_SEMA42_Busy (*C enumerator*), 651
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_SEMA42_Reseting (*C enumerator*), 651
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_SPI_BaudrateNotSupport (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_SPI_Busy (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_SPI_Error (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_SPI_Idle (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_SPI_Timeout (*C enumerator*), 663
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_StartNotAllowed (*C enumerator*), 614
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_Success (*C enumerator*), 502
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_Timeout (*C enumerator*), 502
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USART_BaudrateNotSupport (*C enumerator*), 687
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USART_FramingError (*C enumerator*), 687
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USART_NoiseError (*C enumerator*), 687
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USART_ParityError (*C enumerator*), 687
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USART_RxBusy (*C enumerator*), 686
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USART_RxError (*C enumerator*), 687
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USART_RxIdle (*C enumerator*), 686
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USART_RxRingBufferOverrun (*C enumerator*), 687
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USART_RxStart (*C enumera-*

tor), 687
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USART_TxBusy (*C enumerator*), 686
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USART_TxError (*C enumerator*), 687
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USART_TxIdle (*C enumerator*), 686
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USDHC_BusyTransferring (*C enumerator*), 706
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USDHC_DMADataAddrNotAlign (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USDHC_NotSupport (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USDHC_PrepareAdmaDescriptorFailed (*C enumerator*), 706
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USDHC_ReTuningRequest (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USDHC_SendCommandFailed (*C enumerator*), 706
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USDHC_SendCommandSuccess (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USDHC_TransferDataComplete (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USDHC_TransferDataFailed (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USDHC_TransferDMAComplete (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatus_USDHC_TuningError (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatusGroup_FlexSPI (*C enumerator*), 444
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatusGroup_FlexSPINOR (*C enumerator*), 444
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kStatusGroup_OtpGroup (*C enumerator*), 460
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma1DescriptorActivity1Flag (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma1DescriptorActivity2Flag (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma1DescriptorEndFlag (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma1DescriptorInterrupFlag (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma1DescriptorTypeLink (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma1DescriptorTypeNop (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma1DescriptorTypeSetLength (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma1DescriptorTypeTransfer (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma1DescriptorValidFlag (*C enumerator*), 718
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma2DescriptorActivity1Flag (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma2DescriptorActivity2Flag (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma2DescriptorEndFlag (*C*

enumerator), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma2DescriptorInterruptFlag (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma2DescriptorTypeLink (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma2DescriptorTypeNop (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma2DescriptorTypeReserved (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma2DescriptorTypeTransfer (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Adma2DescriptorValidFlag (*C enumerator*), 719
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AdmaDescriptorErrorFlag (*C enumerator*), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AdmaDescriptorMultipleFlag (*C enumerator*), 720
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AdmaDescriptorSingleFlag (*C enumerator*), 720
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AdmaErrorState (*C enumerator*), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AdmaErrorStateChangeAddress (*C enumerator*), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AdmaErrorStateFetchDescriptor (*C enumerator*), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AdmaErrorStateInvalidDescriptor (*C enumerator*), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AdmaErrorStateInvalidLength (*C enumerator*), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AdmaErrorStateStopDma (*C enumerator*), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AdmaErrorStateTransferData (*C enumerator*), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AdmaLenghMismatchFlag (*C enumerator*), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AllInterruptFlags (*C enumerator*), 713
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AutoCommand12CrcErrorFlag (*C enumerator*), 713
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AutoCommand12EndBitErrorFlag (*C enumerator*), 713
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AutoCommand12ErrorFlag (*C enumerator*), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AutoCommand12IndexErrorFlag (*C enumerator*), 713
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AutoCommand12NotExecutedFlag (*C enumerator*), 713
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AutoCommand12NotIssuedFlag (*C enumerator*), 713
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_AutoCommand12TimeoutFlag (*C enumerator*), 713
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_BlockGapEventFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_BufferReadEnableFlag (*C enumerator*), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_BufferReadReadyFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_BufferWriteEnableFlag (*C enu-*

merator), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_BufferWriteReadyFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CardDetectFlag (*C enumerator*), 713
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CardInsertedFlag (*C enumerator*), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CardInsertionFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CardInterruptFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CardRemovalFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CommandCompleteFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CommandCrcErrorFlag (*C enumerator*), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CommandEndBitErrorFlag (*C enumerator*), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CommandErrorFlag (*C enumerator*), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CommandFlag (*C enumerator*), 713
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CommandIndexErrorFlag (*C enumerator*), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CommandInhibitFlag (*C enumerator*), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CommandLineLevelFlag (*C enumerator*), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CommandTimeoutFlag (*C enumerator*), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CommandTypeAbortFlag (*C enumerator*), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CommandTypeResumeFlag (*C enumerator*), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_CommandTypeSuspendFlag (*C enumerator*), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Data0LineLevelFlag (*C enumerator*), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Data1LineLevelFlag (*C enumerator*), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Data2LineLevelFlag (*C enumerator*), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Data3LineLevelFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Data4LineLevelFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Data5LineLevelFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Data6LineLevelFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Data7LineLevelFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DataCompleteFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DataCrcErrorFlag (*C enumerator*), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DataDMAFlag (*C enumera-*

tor), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DataEndBitErrorFlag (C enumerator), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DataErrorFlag (C enumerator), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DataFlag (C enumerator), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DataInhibitFlag (C enumerator), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DataLineActiveFlag (C enumerator), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DataPresentFlag (C enumerator), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DataReadFlag (C enumerator), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DataTimeoutFlag (C enumerator), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DelaySettingFinishedFlag (C enumerator), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DmaCompleteFlag (C enumerator), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_DmaErrorFlag (C enumerator), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_EnableAutoCommand12Flag (C enumerator), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_EnableAutoCommand23Flag (C enumerator), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_EnableBlockCountFlag (C enumerator), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_EnableCrcCheckFlag (C enumerator), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_EnableDmaFlag (C enumerator), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_EnableIndexCheckFlag (C enumerator), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ErrorFlag (C enumerator), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ExactBlockNumberReadFlag (C enumerator), 717
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ExecuteTuning (C enumerator), 713
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12CrcError (C enumerator), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12Error (C enumerator), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12IndexError (C enumerator), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12NotExecuted (C enumerator), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12NotIssued (C enumerator), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventAutoCommand12Timeout (C enumerator), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventCardInt (C enumerator), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventCommandCrcError (C enumerator), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventCommandEndBitError (C enumerator), 715

drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventCommandIndexError (*C enumerator*), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventCommandTimeout (*C enumerator*), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventDataCrcError (*C enumerator*), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventDataEndBitError (*C enumerator*), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventDataTimeout (*C enumerator*), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventDmaError (*C enumerator*), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventEndBitError (*C enumerator*), 714
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventsAll (*C enumerator*), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ForceEventTuningError (*C enumerator*), 715
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_InterruptAtBlockGapFlag (*C enumerator*), 717
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_MultipleBlockFlag (*C enumerator*), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ReadDoneNo8CLK (*C enumerator*), 717
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ReadTransferActiveFlag (*C enumerator*), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ReadWaitControlFlag (*C enumerator*), 717
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ResetAll (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ResetCommand (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ResetData (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ResetsAll (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ResetTuning (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ResponseLength48BusyFlag (*C enumerator*), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ResponseLength48Flag (*C enumerator*), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ResponseLength136Flag (*C enumerator*), 709
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ReTuningEventFlag (*C enumerator*), 711
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_ReTuningRequestFlag (*C enumerator*), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_SdClockStableFlag (*C enumerator*), 710
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_SDR104TuningFlag (*C enumerator*), 713
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_StopAtBlockGapFlag (*C enumerator*), 717
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Support4BitFlag (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_Support8BitFlag (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_SupportAdmaFlag (*C enumerator*), 707

drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_SupportDDR50Flag (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_SupportDmaFlag (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_SupportHighSpeedFlag (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_SupportSDR50Flag (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_SupportSDR104Flag (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_SupportSuspendResumeFlag (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_SupportV300Flag (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_SupportV330Flag (*C enumerator*), 707
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_TransferDataBoot (*C enumerator*), 720
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_TransferDataBootcontinous (*C enumerator*), 720
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_TransferDataNormal (*C enumerator*), 720
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_TransferDataTuning (*C enumerator*), 720
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_TuningErrorFlag (*C enumerator*), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_TuningPassFlag (*C enumerator*), 712
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_TuningSampleClockSel (*C enumerator*), 713
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_WakeupEventOnCardInsert (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_WakeupEventOnCardInt (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_WakeupEventOnCardRemove (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_WakeupEventsAll (*C enumerator*), 708
drivers_RT_RT600_MIMXRT685S.PhonyNameDueToError.kUSDHC_WriteTransferActiveFlag (*C enumerator*), 710
drivers_RT_RT600_MIMXRT685S.PININT_BITSLICE_CFG_MASK (*C macro*), 563
drivers_RT_RT600_MIMXRT685S.PININT_BITSLICE_CFG_START (*C macro*), 563
drivers_RT_RT600_MIMXRT685S.PININT_BITSLICE_ENDP_MASK (*C macro*), 563
drivers_RT_RT600_MIMXRT685S.PININT_BITSLICE_SRC_MASK (*C macro*), 562
drivers_RT_RT600_MIMXRT685S.PININT_BITSLICE_SRC_START (*C macro*), 562
drivers_RT_RT600_MIMXRT685S.pint_cb_t (*C type*), 556
drivers_RT_RT600_MIMXRT685S.PINT_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.PINT_Deinit (*C function*), 561
drivers_RT_RT600_MIMXRT685S.PINT_DisableCallback (*C function*), 561
drivers_RT_RT600_MIMXRT685S.PINT_DisableCallbackByIndex (*C function*), 562
drivers_RT_RT600_MIMXRT685S.PINT_DisableInterruptByIndex (*C function*), 562
drivers_RT_RT600_MIMXRT685S.PINT_DriverIRQHandler (*C function*), 562
drivers_RT_RT600_MIMXRT685S.PINT_EnableCallback (*C function*), 561
drivers_RT_RT600_MIMXRT685S.PINT_EnableCallbackByIndex (*C function*), 561
drivers_RT_RT600_MIMXRT685S.PINT_EnableInterruptByIndex (*C function*), 562
drivers_RT_RT600_MIMXRT685S.PINT_Init (*C function*), 556
drivers_RT_RT600_MIMXRT685S.PINT_PatternMatchConfig (*C function*), 559
drivers_RT_RT600_MIMXRT685S.PINT_PatternMatchDisable (*C function*), 560

drivers_RT_RT600_MIMXRT685S.PINT_PatternMatchDisableRXEV (*C function*), 561
drivers_RT_RT600_MIMXRT685S.PINT_PatternMatchEnable (*C function*), 560
drivers_RT_RT600_MIMXRT685S.PINT_PatternMatchEnableRXEV (*C function*), 560
drivers_RT_RT600_MIMXRT685S.PINT_PatternMatchGetConfig (*C function*), 559
drivers_RT_RT600_MIMXRT685S.PINT_PatternMatchGetStatus (*C function*), 560
drivers_RT_RT600_MIMXRT685S.PINT_PatternMatchGetStatusAll (*C function*), 560
drivers_RT_RT600_MIMXRT685S.PINT_PatternMatchResetDetectLogic (*C function*), 560
drivers_RT_RT600_MIMXRT685S.PINT_PIN_BOTH_EDGE (*C macro*), 563
drivers_RT_RT600_MIMXRT685S.pint_pin_enable_t (*C type*), 556
drivers_RT_RT600_MIMXRT685S.PINT_PIN_FALL_EDGE (*C macro*), 563
drivers_RT_RT600_MIMXRT685S.PINT_PIN_HIGH_LEVEL (*C macro*), 563
drivers_RT_RT600_MIMXRT685S.PINT_PIN_INT_EDGE (*C macro*), 563
drivers_RT_RT600_MIMXRT685S.PINT_PIN_INT_FALL_OR_HIGH_LEVEL (*C macro*), 563
drivers_RT_RT600_MIMXRT685S.PINT_PIN_INT_HIGH_OR_RISE_TRIGGER (*C macro*), 530
drivers_RT_RT600_MIMXRT685S.PINT_PIN_INT_LEVEL (*C macro*), 563
drivers_RT_RT600_MIMXRT685S.PINT_PIN_INT_LOW_OR_FALL_TRIGGER (*C macro*), 530
drivers_RT_RT600_MIMXRT685S.PINT_PIN_INT_RISE (*C macro*), 563
drivers_RT_RT600_MIMXRT685S.pint_pin_int_t (*C type*), 556
drivers_RT_RT600_MIMXRT685S.PINT_PIN_LOW_LEVEL (*C macro*), 563
drivers_RT_RT600_MIMXRT685S.PINT_PIN_RISE_EDGE (*C macro*), 563
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptClrFallFlag (*C function*), 558
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptClrFallFlagAll (*C function*), 558
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptClrRiseFlag (*C function*), 558
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptClrRiseFlagAll (*C function*), 559
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptClrStatus (*C function*), 557
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptClrStatusAll (*C function*), 557
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptConfig (*C function*), 556
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptGetConfig (*C function*), 557
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptGetFallFlag (*C function*), 558
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptGetFallFlagAll (*C function*), 558
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptGetRiseFlag (*C function*), 558
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptGetRiseFlagAll (*C function*), 559
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptGetStatus (*C function*), 557
drivers_RT_RT600_MIMXRT685S.PINT_PinInterruptGetStatusAll (*C function*), 557
drivers_RT_RT600_MIMXRT685S.pint_pmatch_bslice_cfg_t (*C type*), 556
drivers_RT_RT600_MIMXRT685S.pint_pmatch_bslice_t (*C type*), 556
drivers_RT_RT600_MIMXRT685S.pint_pmatch_cfg_t (*C type*), 556
drivers_RT_RT600_MIMXRT685S.pint_pmatch_input_src_t (*C type*), 556
drivers_RT_RT600_MIMXRT685S.PINT_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.PINT_SetCallback (*C function*), 556
drivers_RT_RT600_MIMXRT685S.pint_status_t (*C type*), 556
drivers_RT_RT600_MIMXRT685S.PINT_USE_LEGACY_CALLBACK (*C macro*), 562
drivers_RT_RT600_MIMXRT685S.PINTSEL_PMUX_ID (*C macro*), 484
drivers_RT_RT600_MIMXRT685S.pmic_mode_reg_t (*C type*), 572
drivers_RT_RT600_MIMXRT685S.PMIC_VDDCORE_RECOVERY_TIME_IGNORE (*C macro*), 578
drivers_RT_RT600_MIMXRT685S.PMUX_SHIFT (*C macro*), 485
drivers_RT_RT600_MIMXRT685S.POWER_ApplyPD (*C function*), 574
drivers_RT_RT600_MIMXRT685S.POWER_ClearEventFlags (*C function*), 574
drivers_RT_RT600_MIMXRT685S.power_control_for_pmic_mode (*C type*), 573
drivers_RT_RT600_MIMXRT685S.POWER_DEFAULT_PMICMODE_DPD (*C macro*), 578
drivers_RT_RT600_MIMXRT685S.POWER_DEFAULT_PMICMODE_DS (*C macro*), 578
drivers_RT_RT600_MIMXRT685S.POWER_DEFAULT_PMICMODE_FDPD (*C macro*), 578
drivers_RT_RT600_MIMXRT685S.POWER_DisableDeepSleep (*C function*), 573
drivers_RT_RT600_MIMXRT685S.POWER_DisableInterrupts (*C function*), 574
drivers_RT_RT600_MIMXRT685S.POWER_DisableLVD (*C function*), 576
drivers_RT_RT600_MIMXRT685S.POWER_DisablePD (*C function*), 573
drivers_RT_RT600_MIMXRT685S.POWER_EnableDeepSleep (*C function*), 573

drivers_RT_RT600_MIMXRT685S.POWER_EnableInterrupts (*C function*), 574
drivers_RT_RT600_MIMXRT685S.POWER_EnablePD (*C function*), 573
drivers_RT_RT600_MIMXRT685S.POWER_EnterDeepPowerDown (*C function*), 577
drivers_RT_RT600_MIMXRT685S.POWER_EnterDeepSleep (*C function*), 576
drivers_RT_RT600_MIMXRT685S.POWER_EnterFbb (*C function*), 575
drivers_RT_RT600_MIMXRT685S.POWER_EnterFullDeepPowerDown (*C function*), 577
drivers_RT_RT600_MIMXRT685S.POWER_EnterNbb (*C function*), 575
drivers_RT_RT600_MIMXRT685S.POWER_EnterPowerMode (*C function*), 577
drivers_RT_RT600_MIMXRT685S.POWER_EnterRbb (*C function*), 575
drivers_RT_RT600_MIMXRT685S.POWER_EnterSleep (*C function*), 576
drivers_RT_RT600_MIMXRT685S.POWER_GetBodyBiasMode (*C function*), 574
drivers_RT_RT600_MIMXRT685S.POWER_GetEventFlags (*C function*), 574
drivers_RT_RT600_MIMXRT685S.POWER_GetLibVersion (*C function*), 577
drivers_RT_RT600_MIMXRT685S.POWER_GetLvdFallingTripVoltage (*C function*), 576
drivers_RT_RT600_MIMXRT685S.POWER_GetPmicMode (*C function*), 574
drivers_RT_RT600_MIMXRT685S.power_lvd_falling_trip_vol_val_t (*C type*), 572
drivers_RT_RT600_MIMXRT685S.power_mode_cfg_t (*C type*), 572
drivers_RT_RT600_MIMXRT685S.power_pad_vrange_t (*C type*), 572
drivers_RT_RT600_MIMXRT685S.power_pad_vrange_val_t (*C type*), 572
drivers_RT_RT600_MIMXRT685S.power_part_temp_range_t (*C type*), 573
drivers_RT_RT600_MIMXRT685S.POWER_PmicPowerModeSelectControl (*C function*), 573
drivers_RT_RT600_MIMXRT685S.POWER_RestoreLVD (*C function*), 576
drivers_RT_RT600_MIMXRT685S.POWER_SetAnalogBuffer (*C function*), 574
drivers_RT_RT600_MIMXRT685S.POWER_SetLdoVoltageForFreq (*C function*), 575
drivers_RT_RT600_MIMXRT685S.POWER_SetLvdFallingTripVoltage (*C function*), 576
drivers_RT_RT600_MIMXRT685S.POWER_SetPadVolRange (*C function*), 575
drivers_RT_RT600_MIMXRT685S.POWER_SetPmicCoreSupplyFunc (*C function*), 575
drivers_RT_RT600_MIMXRT685S.POWER_SetPmicMode (*C function*), 576
drivers_RT_RT600_MIMXRT685S.POWER_SetVddCoreSupplySrc (*C function*), 575
drivers_RT_RT600_MIMXRT685S.POWER_SetVoltageForFreq (*C function*), 575
drivers_RT_RT600_MIMXRT685S.POWER_UpdateOscSettlingTime (*C function*), 573
drivers_RT_RT600_MIMXRT685S.POWER_UpdatePmicRecoveryTime (*C function*), 574
drivers_RT_RT600_MIMXRT685S.power_vddcore_set_func_t (*C type*), 573
drivers_RT_RT600_MIMXRT685S.power_vddcore_src_t (*C type*), 573
drivers_RT_RT600_MIMXRT685S.power_volt_op_range_t (*C type*), 573
drivers_RT_RT600_MIMXRT685S.POWERQUAD_CLOCKS (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.POWERQUAD_MAKE_MATRIX_LEN (*C macro*), 607
drivers_RT_RT600_MIMXRT685S.POWERQUAD_RSTS (*C macro*), 621
drivers_RT_RT600_MIMXRT685S.PQ_Arctan2Fixed (*C function*), 588
drivers_RT_RT600_MIMXRT685S.PQ_ArctanFixed (*C function*), 587
drivers_RT_RT600_MIMXRT685S.PQ_ArctanhFixed (*C function*), 588
drivers_RT_RT600_MIMXRT685S.pq_arm_cfft_q15 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_cfft_q31 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_conv_f32 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_conv_q15 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_conv_q31 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_correlate_f32 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_correlate_q15 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_correlate_q31 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_cos_f32 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_cos_q15 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_cos_q31 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_dct4_init_q15 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_dct4_init_q31 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_dct4_q15 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_dct4_q31 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_fir_f32 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_fir_init_f32 (*C macro*), 608

drivers_RT_RT600_MIMXRT685S.pq_arm_fir_init_q15 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_fir_init_q31 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_fir_q15 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_fir_q31 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_add_f32 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_add_q15 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_add_q31 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_init_f32 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_init_q15 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_init_q31 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_inverse_f32 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_mult_f32 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_mult_q15 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_mult_q31 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_scale_f32 (*C macro*), 610
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_scale_q15 (*C macro*), 610
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_scale_q31 (*C macro*), 610
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_sub_f32 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_sub_q15 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_sub_q31 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_trans_f32 (*C macro*), 609
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_trans_q15 (*C macro*), 610
drivers_RT_RT600_MIMXRT685S.pq_arm_mat_trans_q31 (*C macro*), 610
drivers_RT_RT600_MIMXRT685S.pq_arm_rfft_init_q15 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_rfft_init_q31 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_rfft_q15 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_rfft_q31 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_sin_f32 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_sin_q15 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_sin_q31 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_sqrt_q15 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.pq_arm_sqrt_q31 (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.PQ_BIQ0_CALC (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_BIQ1_CALC (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_BIQUAD (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_Biquad1F32 (*C function*), 581
drivers_RT_RT600_MIMXRT685S.PQ_Biquad1Fixed (*C function*), 589
drivers_RT_RT600_MIMXRT685S.pq_biquad_cascade_df2_instance (*C struct*), 612
drivers_RT_RT600_MIMXRT685S.pq_biquad_cascade_df2_instance.numStages (*C var*), 612
drivers_RT_RT600_MIMXRT685S.pq_biquad_cascade_df2_instance.pState (*C var*), 612
drivers_RT_RT600_MIMXRT685S.PQ_BIQUAD_FIXED (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.pq_biquad_param_t (*C type*), 595
drivers_RT_RT600_MIMXRT685S.pq_biquad_state_t (*C type*), 595
drivers_RT_RT600_MIMXRT685S.PQ_BiquadBackUpInternalState (*C function*), 590
drivers_RT_RT600_MIMXRT685S.PQ_BiquadCascadeDf2F32 (*C function*), 591
drivers_RT_RT600_MIMXRT685S.PQ_BiquadCascadeDf2Fixed16 (*C function*), 591
drivers_RT_RT600_MIMXRT685S.PQ_BiquadCascadeDf2Fixed32 (*C function*), 591
drivers_RT_RT600_MIMXRT685S.PQ_BiquadCascadeDf2Init (*C function*), 590
drivers_RT_RT600_MIMXRT685S.PQ_BiquadF32 (*C function*), 581
drivers_RT_RT600_MIMXRT685S.PQ_BiquadFixed (*C function*), 583
drivers_RT_RT600_MIMXRT685S.PQ_BiquadRestoreInternalState (*C function*), 590
drivers_RT_RT600_MIMXRT685S.PQ_COMP0_ONLY (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_COMP1_ONLY (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.pq_computationengine_t (*C enum*), 594
drivers_RT_RT600_MIMXRT685S.pq_computationengine_t.kPQ_CP_CORDIC (*C enumerator*), 594
drivers_RT_RT600_MIMXRT685S.pq_computationengine_t.kPQ_CP_FFT (*C enumerator*), 594
drivers_RT_RT600_MIMXRT685S.pq_computationengine_t.kPQ_CP_FIR (*C enumerator*), 594

drivers_RT_RT600_MIMXRT685S.pq_computationengine_t.kPQ_CP_MTX (*C enumerator*), 594
drivers_RT_RT600_MIMXRT685S.pq_computationengine_t.kPQ_CP_PQ (*C enumerator*), 594
drivers_RT_RT600_MIMXRT685S.pq_config_t (*C struct*), 610
drivers_RT_RT600_MIMXRT685S.pq_config_t.inputAFormat (*C var*), 610
drivers_RT_RT600_MIMXRT685S.pq_config_t.inputAPrescale (*C var*), 610
drivers_RT_RT600_MIMXRT685S.pq_config_t.inputBFormat (*C var*), 610
drivers_RT_RT600_MIMXRT685S.pq_config_t.inputBPrescale (*C var*), 610
drivers_RT_RT600_MIMXRT685S.pq_config_t.machineFormat (*C var*), 611
drivers_RT_RT600_MIMXRT685S.pq_config_t.outputFormat (*C var*), 610
drivers_RT_RT600_MIMXRT685S.pq_config_t.outputPrescale (*C var*), 611
drivers_RT_RT600_MIMXRT685S.pq_config_t.tmpBase (*C var*), 611
drivers_RT_RT600_MIMXRT685S.pq_config_t.tmpFormat (*C var*), 611
drivers_RT_RT600_MIMXRT685S.pq_config_t.tmpPrescale (*C var*), 611
drivers_RT_RT600_MIMXRT685S.pq_cordic_iter_t (*C enum*), 595
drivers_RT_RT600_MIMXRT685S.pq_cordic_iter_t.kPQ_Iteration_8 (*C enumerator*), 595
drivers_RT_RT600_MIMXRT685S.pq_cordic_iter_t.kPQ_Iteration_16 (*C enumerator*), 595
drivers_RT_RT600_MIMXRT685S.pq_cordic_iter_t.kPQ_Iteration_24 (*C enumerator*), 595
drivers_RT_RT600_MIMXRT685S.PQ_COS (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_COS_INF (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_CosF32 (*C function*), 581
drivers_RT_RT600_MIMXRT685S.PQ_CosQ15 (*C function*), 583
drivers_RT_RT600_MIMXRT685S.PQ_CosQ31 (*C function*), 582
drivers_RT_RT600_MIMXRT685S.PQ_Deinit (*C function*), 579
drivers_RT_RT600_MIMXRT685S.PQ_DF2_Cascade_Vector8_FP (*C macro*), 604
drivers_RT_RT600_MIMXRT685S.PQ_DF2_Cascade_Vector8_FX (*C macro*), 605
drivers_RT_RT600_MIMXRT685S.PQ_DF2_Vector8_FP (*C macro*), 602
drivers_RT_RT600_MIMXRT685S.PQ_DF2_Vector8_FX (*C macro*), 602
drivers_RT_RT600_MIMXRT685S.PQ_DIV (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_DivF32 (*C function*), 581
drivers_RT_RT600_MIMXRT685S.PQ_End_Vector_Func (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_EndVector (*C macro*), 601
drivers_RT_RT600_MIMXRT685S.PQ_ERRSTAT_FIXEDOVERFLOW (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_ERRSTAT_NAN (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_ERRSTAT_OVERFLOW (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_ERRSTAT_UNDERFLOW (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_ETONX (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_ETONX_INF (*C macro*), 599
drivers_RT_RT600_MIMXRT685S.PQ_EtonxF32 (*C function*), 580
drivers_RT_RT600_MIMXRT685S.PQ_EtonxFixed (*C function*), 582
drivers_RT_RT600_MIMXRT685S.PQ_ETOX (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_ETOX_INF (*C macro*), 599
drivers_RT_RT600_MIMXRT685S.PQ_EtoxF32 (*C function*), 580
drivers_RT_RT600_MIMXRT685S.PQ_EtoxFixed (*C function*), 582
drivers_RT_RT600_MIMXRT685S.PQ_FIR (*C function*), 591
drivers_RT_RT600_MIMXRT685S.PQ_FIR_CONVOLUTION (*C macro*), 598
drivers_RT_RT600_MIMXRT685S.PQ_FIR_CORRELATION (*C macro*), 598
drivers_RT_RT600_MIMXRT685S.PQ_FIR_FIR (*C macro*), 598
drivers_RT_RT600_MIMXRT685S.PQ_FIR_INCREMENTAL (*C macro*), 598
drivers_RT_RT600_MIMXRT685S.PQ_FIRIncrement (*C function*), 592
drivers_RT_RT600_MIMXRT685S.PQ_FIXEDPT (*C macro*), 595
drivers_RT_RT600_MIMXRT685S.PQ_FLOAT32 (*C macro*), 595
drivers_RT_RT600_MIMXRT685S.pq_float_t (*C type*), 595
drivers_RT_RT600_MIMXRT685S.pq_format_t (*C enum*), 594
drivers_RT_RT600_MIMXRT685S.pq_format_t.kPQ_16Bit (*C enumerator*), 594
drivers_RT_RT600_MIMXRT685S.pq_format_t.kPQ_32Bit (*C enumerator*), 594
drivers_RT_RT600_MIMXRT685S.pq_format_t.kPQ_Float (*C enumerator*), 595
drivers_RT_RT600_MIMXRT685S.PQ_GetDefaultConfig (*C function*), 579
drivers_RT_RT600_MIMXRT685S.PQ_Init (*C function*), 579

drivers_RT_RT600_MIMXRT685S.PQ_Initiate_Vector_Func (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_INV (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_INV_INF (*C macro*), 599
drivers_RT_RT600_MIMXRT685S.PQ_InvF32 (*C function*), 580
drivers_RT_RT600_MIMXRT685S.PQ_InvFixed (*C function*), 581
drivers_RT_RT600_MIMXRT685S.PQ_INVSQRT (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_InvSqrtF32 (*C function*), 580
drivers_RT_RT600_MIMXRT685S.PQ_InvSqrtFixed (*C function*), 582
drivers_RT_RT600_MIMXRT685S.PQ_ISQRT_INF (*C macro*), 599
drivers_RT_RT600_MIMXRT685S.PQ_LN (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_LN_INF (*C macro*), 599
drivers_RT_RT600_MIMXRT685S.PQ_LnF32 (*C function*), 580
drivers_RT_RT600_MIMXRT685S.PQ_LnFixed (*C function*), 581
drivers_RT_RT600_MIMXRT685S.PQ_MatrixAddition (*C function*), 592
drivers_RT_RT600_MIMXRT685S.PQ_MatrixInversion (*C function*), 593
drivers_RT_RT600_MIMXRT685S.PQ_MatrixMultiplication (*C function*), 592
drivers_RT_RT600_MIMXRT685S.PQ_MatrixProduct (*C function*), 593
drivers_RT_RT600_MIMXRT685S.PQ_MatrixScale (*C function*), 594
drivers_RT_RT600_MIMXRT685S.PQ_MatrixSubtraction (*C function*), 592
drivers_RT_RT600_MIMXRT685S.PQ_MatrixTranspose (*C function*), 593
drivers_RT_RT600_MIMXRT685S.PQ_MTX_ADD (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_MTX_INV (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_MTX_MULT (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_MTX_PROD (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_MTX_SCALE (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_MTX_SUB (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_MTX_TRAN (*C macro*), 598
drivers_RT_RT600_MIMXRT685S.pq_prescale_t (*C struct*), 610
drivers_RT_RT600_MIMXRT685S.pq_prescale_t.inputPrescale (*C var*), 610
drivers_RT_RT600_MIMXRT685S.pq_prescale_t.outputPrescale (*C var*), 610
drivers_RT_RT600_MIMXRT685S.pq_prescale_t.outputSaturate (*C var*), 610
drivers_RT_RT600_MIMXRT685S.PQ_Q15_2_FLOAT (*C macro*), 608
drivers_RT_RT600_MIMXRT685S.PQ_Q31_2_FLOAT (*C macro*), 607
drivers_RT_RT600_MIMXRT685S.PQ_RUN_OPCODE_R2_R3 (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_RUN_OPCODE_R3_R2 (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_RUN_OPCODE_R4_R5 (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_RUN_OPCODE_R5_R4 (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_RUN_OPCODE_R6_R7 (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_RUN_OPCODE_R7_R6 (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_SetConfig (*C function*), 579
drivers_RT_RT600_MIMXRT685S.PQ_SetCoprocessorScaler (*C function*), 579
drivers_RT_RT600_MIMXRT685S.PQ_SetFormat (*C function*), 579
drivers_RT_RT600_MIMXRT685S.PQ_SIN (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_SIN_INF (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_SinF32 (*C function*), 580
drivers_RT_RT600_MIMXRT685S.PQ_SinQ15 (*C function*), 582
drivers_RT_RT600_MIMXRT685S.PQ_SinQ31 (*C function*), 582
drivers_RT_RT600_MIMXRT685S.PQ_SQRT (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_SQRT_INF (*C macro*), 599
drivers_RT_RT600_MIMXRT685S.PQ_SqrtF32 (*C function*), 580
drivers_RT_RT600_MIMXRT685S.PQ_SqrtFixed (*C function*), 581
drivers_RT_RT600_MIMXRT685S.PQ_StartVector (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_StartVectorFixed16 (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_StartVectorQ15 (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_TRANS (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_TRANS_CDCT (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_TRANS_CFFT (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_TRANS_FIXED (*C macro*), 596

drivers_RT_RT600_MIMXRT685S.PQ_TRANS_IDCT (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_TRANS_IFFT (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_TRANS_RDCT (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_TRANS_RFFT (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_TransformCDCT (*C function*), 590
drivers_RT_RT600_MIMXRT685S.PQ_TransformCFFT (*C function*), 589
drivers_RT_RT600_MIMXRT685S.PQ_TransformIDCT (*C function*), 590
drivers_RT_RT600_MIMXRT685S.PQ_TransformIFFT (*C function*), 589
drivers_RT_RT600_MIMXRT685S.PQ_TransformRDCT (*C function*), 590
drivers_RT_RT600_MIMXRT685S.PQ_TransformRFFT (*C function*), 589
drivers_RT_RT600_MIMXRT685S.PQ_TRIG (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_TRIG_FIXED (*C macro*), 596
drivers_RT_RT600_MIMXRT685S.PQ_VEC_DOTP (*C macro*), 597
drivers_RT_RT600_MIMXRT685S.PQ_Vector8_FP (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_Vector8_FX (*C macro*), 600
drivers_RT_RT600_MIMXRT685S.PQ_Vector8BiqaudDf2CascadeF32 (*C macro*), 595
drivers_RT_RT600_MIMXRT685S.PQ_Vector8BiqaudDf2CascadeFixed16 (*C macro*), 595
drivers_RT_RT600_MIMXRT685S.PQ_Vector8BiqaudDf2CascadeFixed32 (*C macro*), 595
drivers_RT_RT600_MIMXRT685S.PQ_Vector8BiquadDf2CascadeF32 (*C macro*), 605
drivers_RT_RT600_MIMXRT685S.PQ_Vector8BiquadDf2CascadeFixed16 (*C macro*), 607
drivers_RT_RT600_MIMXRT685S.PQ_Vector8BiquadDf2CascadeFixed32 (*C macro*), 606
drivers_RT_RT600_MIMXRT685S.PQ_Vector8BiquadDf2F32 (*C macro*), 603
drivers_RT_RT600_MIMXRT685S.PQ_Vector8BiquadDf2Fixed16 (*C macro*), 604
drivers_RT_RT600_MIMXRT685S.PQ_Vector8BiquadDf2Fixed32 (*C macro*), 603
drivers_RT_RT600_MIMXRT685S.PQ_Vector8F32 (*C macro*), 601
drivers_RT_RT600_MIMXRT685S.PQ_Vector8Fixed16 (*C macro*), 601
drivers_RT_RT600_MIMXRT685S.PQ_Vector8Fixed32 (*C macro*), 601
drivers_RT_RT600_MIMXRT685S.PQ_Vector8Q15 (*C macro*), 602
drivers_RT_RT600_MIMXRT685S.PQ_VectorBiqaudCascadeDf2F32 (*C macro*), 595
drivers_RT_RT600_MIMXRT685S.PQ_VectorBiqaudCascadeDf2Fixed16 (*C macro*), 595
drivers_RT_RT600_MIMXRT685S.PQ_VectorBiqaudCascadeDf2Fixed32 (*C macro*), 595
drivers_RT_RT600_MIMXRT685S.PQ_VectorBiqaudDf2F32 (*C macro*), 595
drivers_RT_RT600_MIMXRT685S.PQ_VectorBiqaudDf2Fixed16 (*C macro*), 595
drivers_RT_RT600_MIMXRT685S.PQ_VectorBiqaudDf2Fixed32 (*C macro*), 595
drivers_RT_RT600_MIMXRT685S.PQ_VectorBiquadCascadeDf2F32 (*C function*), 587
drivers_RT_RT600_MIMXRT685S.PQ_VectorBiquadCascadeDf2Fixed16 (*C function*), 587
drivers_RT_RT600_MIMXRT685S.PQ_VectorBiquadCascadeDf2Fixed32 (*C function*), 587
drivers_RT_RT600_MIMXRT685S.PQ_VectorBiquadDf2F32 (*C function*), 587
drivers_RT_RT600_MIMXRT685S.PQ_VectorBiquadDf2Fixed16 (*C function*), 587
drivers_RT_RT600_MIMXRT685S.PQ_VectorBiquadDf2Fixed32 (*C function*), 587
drivers_RT_RT600_MIMXRT685S.PQ_VectorCosF32 (*C function*), 584
drivers_RT_RT600_MIMXRT685S.PQ_VectorCosQ15 (*C function*), 585
drivers_RT_RT600_MIMXRT685S.PQ_VectorCosQ31 (*C function*), 585
drivers_RT_RT600_MIMXRT685S.PQ_VectorDotProduct (*C function*), 593
drivers_RT_RT600_MIMXRT685S.PQ_VectorEtonxF32 (*C function*), 584
drivers_RT_RT600_MIMXRT685S.PQ_VectorEtonxFixed16 (*C function*), 586
drivers_RT_RT600_MIMXRT685S.PQ_VectorEtonxFixed32 (*C function*), 585
drivers_RT_RT600_MIMXRT685S.PQ_VectorEtoxF32 (*C function*), 583
drivers_RT_RT600_MIMXRT685S.PQ_VectorEtoxFixed16 (*C function*), 586
drivers_RT_RT600_MIMXRT685S.PQ_VectorEtoxFixed32 (*C function*), 585
drivers_RT_RT600_MIMXRT685S.PQ_VectorInvF32 (*C function*), 583
drivers_RT_RT600_MIMXRT685S.PQ_VectorInvFixed16 (*C function*), 586
drivers_RT_RT600_MIMXRT685S.PQ_VectorInvFixed32 (*C function*), 584
drivers_RT_RT600_MIMXRT685S.PQ_VectorInvSqrtF32 (*C function*), 583
drivers_RT_RT600_MIMXRT685S.PQ_VectorInvSqrtFixed16 (*C function*), 586
drivers_RT_RT600_MIMXRT685S.PQ_VectorInvSqrtFixed32 (*C function*), 585
drivers_RT_RT600_MIMXRT685S.PQ_VectorLnF32 (*C function*), 583
drivers_RT_RT600_MIMXRT685S.PQ_VectorLnFixed16 (*C function*), 586

drivers_RT_RT600_MIMXRT685S.PQ_VectorLnFixed32 (*C function*), 584
drivers_RT_RT600_MIMXRT685S.PQ_VectorSinF32 (*C function*), 584
drivers_RT_RT600_MIMXRT685S.PQ_VectorSinQ15 (*C function*), 585
drivers_RT_RT600_MIMXRT685S.PQ_VectorSinQ31 (*C function*), 585
drivers_RT_RT600_MIMXRT685S.PQ_VectorSqrtF32 (*C function*), 583
drivers_RT_RT600_MIMXRT685S.PQ_VectorSqrtFixed16 (*C function*), 586
drivers_RT_RT600_MIMXRT685S.PQ_VectorSqrtFixed32 (*C function*), 584
drivers_RT_RT600_MIMXRT685S.PQ_WaitDone (*C function*), 579
drivers_RT_RT600_MIMXRT685S.PUF_ACTIVATION_CODE_SIZE (*C macro*), 615
drivers_RT_RT600_MIMXRT685S.PUF_CLOCKS (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.puf_config_t (*C struct*), 615
drivers_RT_RT600_MIMXRT685S.PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE (*C macro*), 615
drivers_RT_RT600_MIMXRT685S.puf_key_index_register_t (*C type*), 614
drivers_RT_RT600_MIMXRT685S.puf_key_slot_t (*C type*), 615
drivers_RT_RT600_MIMXRT685S.PUF_MIN_KEY_CODE_SIZE (*C macro*), 615
drivers_RT_RT600_MIMXRT685S.puf_min_max_t (*C type*), 615
drivers_RT_RT600_MIMXRT685S.PUF_RSTS (*C macro*), 621
drivers_RT_RT600_MIMXRT685S.RAMFUNCTION_SECTION_CODE (*C macro*), 493
drivers_RT_RT600_MIMXRT685S.RESET_ClearPeripheralReset (*C function*), 619
drivers_RT_RT600_MIMXRT685S.reset_ip_name_t (*C type*), 619
drivers_RT_RT600_MIMXRT685S.RESET_PeripheralReset (*C function*), 619
drivers_RT_RT600_MIMXRT685S.RESET_ReleasePeripheralReset (*C function*), 619
drivers_RT_RT600_MIMXRT685S.RESET_SetPeripheralReset (*C function*), 619
drivers_RT_RT600_MIMXRT685S.RST_CTL0_PSCCTL0 (*C macro*), 619
drivers_RT_RT600_MIMXRT685S.RST_CTL0_PSCCTL1 (*C macro*), 619
drivers_RT_RT600_MIMXRT685S.RST_CTL0_PSCCTL2 (*C macro*), 619
drivers_RT_RT600_MIMXRT685S.RST_CTL1_PSCCTL0 (*C macro*), 619
drivers_RT_RT600_MIMXRT685S.RST_CTL1_PSCCTL1 (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.RST_CTL1_PSCCTL2 (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.RSTCTL_RSTn_t (*C type*), 619
drivers_RT_RT600_MIMXRT685S.RTC_ClearStatusFlags (*C function*), 624
drivers_RT_RT600_MIMXRT685S.RTC_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.rtc_datetime_t (*C type*), 625
drivers_RT_RT600_MIMXRT685S.RTC_Deinit (*C function*), 621
drivers_RT_RT600_MIMXRT685S.RTC_DisableInterrupts (*C function*), 623
drivers_RT_RT600_MIMXRT685S.RTC_EnableAlarmTimerInterruptFromDPD (*C function*), 623
drivers_RT_RT600_MIMXRT685S.RTC_EnableInterrupts (*C function*), 623
drivers_RT_RT600_MIMXRT685S.RTC_EnableSubsecCounter (*C function*), 622
drivers_RT_RT600_MIMXRT685S.RTC_EnableTimer (*C function*), 624
drivers_RT_RT600_MIMXRT685S.RTC_EnableWakeupTimer (*C function*), 622
drivers_RT_RT600_MIMXRT685S.RTC_EnableWakeUpTimerInterruptFromDPD (*C function*), 623
drivers_RT_RT600_MIMXRT685S.RTC_GetAlarm (*C function*), 622
drivers_RT_RT600_MIMXRT685S.RTC_GetDatetime (*C function*), 621
drivers_RT_RT600_MIMXRT685S.RTC_GetEnabledInterrupts (*C function*), 624
drivers_RT_RT600_MIMXRT685S.RTC_GetEnabledWakeupTimer (*C function*), 622
drivers_RT_RT600_MIMXRT685S.RTC_GetSecondsTimerCount (*C function*), 626
drivers_RT_RT600_MIMXRT685S.RTC_GetSecondsTimerMatch (*C function*), 626
drivers_RT_RT600_MIMXRT685S.RTC_GetStatusFlags (*C function*), 624
drivers_RT_RT600_MIMXRT685S.RTC_GetSubsecValue (*C function*), 623
drivers_RT_RT600_MIMXRT685S.RTC_GetWakeupCount (*C function*), 626
drivers_RT_RT600_MIMXRT685S.RTC_Init (*C function*), 621
drivers_RT_RT600_MIMXRT685S.rtc_interrupt_enable_t (*C type*), 625
drivers_RT_RT600_MIMXRT685S.RTC_Reset (*C function*), 626
drivers_RT_RT600_MIMXRT685S.RTC_SetAlarm (*C function*), 622
drivers_RT_RT600_MIMXRT685S.RTC_SetDatetime (*C function*), 621
drivers_RT_RT600_MIMXRT685S.RTC_SetSecondsTimerCount (*C function*), 626
drivers_RT_RT600_MIMXRT685S.RTC_SetSecondsTimerMatch (*C function*), 626

drivers_RT_RT600_MIMXRT685S.RTC_SetWakeupCount (*C function*), 626
drivers_RT_RT600_MIMXRT685S.RTC_StartTimer (*C function*), 624
drivers_RT_RT600_MIMXRT685S.rtc_status_flags_t (*C type*), 625
drivers_RT_RT600_MIMXRT685S.RTC_StopTimer (*C function*), 625
drivers_RT_RT600_MIMXRT685S.rxData (*C var*), 394
drivers_RT_RT600_MIMXRT685S.rxDataSize (*C var*), 395
drivers_RT_RT600_MIMXRT685S.rxDmaHandle (*C var*), 394, 395
drivers_RT_RT600_MIMXRT685S.s_dummyData (*C var*), 665
drivers_RT_RT600_MIMXRT685S.SCT0_PMUX_ID (*C macro*), 484
drivers_RT_RT600_MIMXRT685S.SCT_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.SCT_EV_STATE_STATEMSKn (*C macro*), 649
drivers_RT_RT600_MIMXRT685S.SCT_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.SCTFCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.SCTFCLKSEL_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.SCTIMER_ClearEventInState (*C function*), 639
drivers_RT_RT600_MIMXRT685S.SCTIMER_ClearStatusFlags (*C function*), 633
drivers_RT_RT600_MIMXRT685S.sctimer_clock_mode_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.sctimer_clock_select_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.sctimer_complementary_pwm_handle_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.sctimer_config_t (*C type*), 649
drivers_RT_RT600_MIMXRT685S.sctimer_conflict_resolution_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.sctimer_counter_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.SCTIMER_CreateAndScheduleEvent (*C function*), 633
drivers_RT_RT600_MIMXRT685S.SCTIMER_Deinit (*C function*), 627
drivers_RT_RT600_MIMXRT685S.SCTIMER_DisableInterrupts (*C function*), 632
drivers_RT_RT600_MIMXRT685S.SCTIMER_DriverIRQHandler (*C function*), 640
drivers_RT_RT600_MIMXRT685S.SCTIMER_EnableInterrupts (*C function*), 632
drivers_RT_RT600_MIMXRT685S.sctimer_event_active_direction_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.sctimer_event_callback_t (*C type*), 649
drivers_RT_RT600_MIMXRT685S.sctimer_event_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.SCTIMER_EventHandleIRQ (*C function*), 640
drivers_RT_RT600_MIMXRT685S.SCTIMER_GetCaptureValue (*C function*), 640
drivers_RT_RT600_MIMXRT685S.SCTIMER_GetCounterState (*C function*), 635
drivers_RT_RT600_MIMXRT685S.SCTIMER_GetCOUNTValue (*C function*), 639
drivers_RT_RT600_MIMXRT685S.SCTIMER_GetCurrentState (*C function*), 634
drivers_RT_RT600_MIMXRT685S.SCTIMER_GetDefaultConfig (*C function*), 627
drivers_RT_RT600_MIMXRT685S.SCTIMER_GetEnabledInterrupts (*C function*), 633
drivers_RT_RT600_MIMXRT685S.SCTIMER_GetEventInState (*C function*), 639
drivers_RT_RT600_MIMXRT685S.SCTIMER_GetStatusFlags (*C function*), 633
drivers_RT_RT600_MIMXRT685S.SCTIMER_IncreaseState (*C function*), 634
drivers_RT_RT600_MIMXRT685S.SCTIMER_Init (*C function*), 627
drivers_RT_RT600_MIMXRT685S.sctimer_input_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.sctimer_interrupt_enable_t (*C type*), 649
drivers_RT_RT600_MIMXRT685S.sctimer_out_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.sctimer_pwm_level_select_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.sctimer_pwm_mode_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.sctimer_pwm_signal_param_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.sctimer_pwm_update_mode_t (*C type*), 648
drivers_RT_RT600_MIMXRT685S.SCTIMER_ScheduleEvent (*C function*), 634
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetCallback (*C function*), 635
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetCounterState (*C function*), 635
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetCOUNTValue (*C function*), 638
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetEventInState (*C function*), 639
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupCaptureAction (*C function*), 635
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupComplementaryPwm (*C function*), 631
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupCounterHaltAction (*C function*), 638
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupCounterLimitAction (*C function*), 637
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupCounterStartAction (*C function*), 638

drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupCounterStopAction (*C function*), 638
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupDmaTriggerAction (*C function*), 638
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupEventActiveDirection (*C function*), 637
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupNextStateAction (*C function*), 636
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupNextStateActionwithLdMethod (*C function*), 636
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupOutputClearAction (*C function*), 637
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupOutputSetAction (*C function*), 637
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupOutputToggleAction (*C function*), 637
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupPwm (*C function*), 628
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupSharedPeriodPwm (*C function*), 629
drivers_RT_RT600_MIMXRT685S.SCTIMER_SetupStateLdMethodAction (*C function*), 636
drivers_RT_RT600_MIMXRT685S.SCTIMER_StartTimer (*C function*), 633
drivers_RT_RT600_MIMXRT685S.sctimer_status_flags_t (*C type*), 649
drivers_RT_RT600_MIMXRT685S.SCTIMER_StopTimer (*C function*), 633
drivers_RT_RT600_MIMXRT685S.SCTIMER_UpdateComplementaryPwmDutycycle (*C function*), 632
drivers_RT_RT600_MIMXRT685S.SCTIMER_UpdatePwmDutycycle (*C function*), 629
drivers_RT_RT600_MIMXRT685S.SCTIMER_UpdatePwmDutycycleByEvent (*C function*), 630
drivers_RT_RT600_MIMXRT685S.SDIO0FCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.SDIO0FCLKSEL_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.SDIO1FCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.SDIO1FCLKSEL_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.SDK_ALIGN (*C macro*), 490, 491
drivers_RT_RT600_MIMXRT685S.SDK_ATOMIC_LOCAL_ADD (*C macro*), 492
drivers_RT_RT600_MIMXRT685S.SDK_ATOMIC_LOCAL_CLEAR (*C macro*), 492
drivers_RT_RT600_MIMXRT685S.SDK_ATOMIC_LOCAL_CLEAR_AND_SET (*C macro*), 492
drivers_RT_RT600_MIMXRT685S.SDK_ATOMIC_LOCAL_COMPARE_AND_SET (*C macro*), 493
drivers_RT_RT600_MIMXRT685S.SDK_ATOMIC_LOCAL_SET (*C macro*), 492
drivers_RT_RT600_MIMXRT685S.SDK_ATOMIC_LOCAL_SUB (*C macro*), 492
drivers_RT_RT600_MIMXRT685S.SDK_ATOMIC_LOCAL_TEST_AND_SET (*C macro*), 493
drivers_RT_RT600_MIMXRT685S.SDK_ATOMIC_LOCAL_TOGGLE (*C macro*), 492
drivers_RT_RT600_MIMXRT685S.SDK_DelayAtLeastUs (*C function*), 503
drivers_RT_RT600_MIMXRT685S.SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY (*C macro*), 256
drivers_RT_RT600_MIMXRT685S.SDK_Free (*C function*), 502
drivers_RT_RT600_MIMXRT685S.SDK_IS_ALIGNED (*C macro*), 491
drivers_RT_RT600_MIMXRT685S.SDK_ISR_EXIT_BARRIER (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.SDK_Malloc (*C function*), 502
drivers_RT_RT600_MIMXRT685S.SDK_SIZEALIGN (*C macro*), 490, 491
drivers_RT_RT600_MIMXRT685S.SDK_SIZEALIGN_DOWN (*C macro*), 491
drivers_RT_RT600_MIMXRT685S.SDK_SIZEALIGN_UP (*C macro*), 491
drivers_RT_RT600_MIMXRT685S.SEMA42_BUSY_POLL_COUNT (*C macro*), 655
drivers_RT_RT600_MIMXRT685S.SEMA42_CLOCKS (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.SEMA42_Deinit (*C function*), 653
drivers_RT_RT600_MIMXRT685S.SEMA42_GATE_NUM_RESET_ALL (*C macro*), 654
drivers_RT_RT600_MIMXRT685S.sema42_gate_status_t (*C type*), 652
drivers_RT_RT600_MIMXRT685S.SEMA42_GATEn (*C macro*), 654
drivers_RT_RT600_MIMXRT685S.SEMA42_GetGateStatus (*C function*), 654
drivers_RT_RT600_MIMXRT685S.SEMA42_Init (*C function*), 652
drivers_RT_RT600_MIMXRT685S.SEMA42_Lock (*C function*), 653
drivers_RT_RT600_MIMXRT685S.SEMA42_ResetAllGates (*C function*), 654
drivers_RT_RT600_MIMXRT685S.SEMA42_ResetGate (*C function*), 654
drivers_RT_RT600_MIMXRT685S.SEMA42_RSTS (*C macro*), 620
drivers_RT_RT600_MIMXRT685S.SEMA42_TryLock (*C function*), 653
drivers_RT_RT600_MIMXRT685S.SEMA42_Unlock (*C function*), 653
drivers_RT_RT600_MIMXRT685S.serial_nor_config_option_t (*C type*), 449

drivers_RT_RT600_MIMXRT685S.sfg_clock_src (*C var*), 263
drivers_RT_RT600_MIMXRT685S.SHSGPIO_PMUX_ID (*C macro*), 484
drivers_RT_RT600_MIMXRT685S.size (*C var*), 309
drivers_RT_RT600_MIMXRT685S.slave2Master (*C var*), 393
drivers_RT_RT600_MIMXRT685S.SPI_ASSERTNUM_SSEL (*C macro*), 665
drivers_RT_RT600_MIMXRT685S.spi_clock_phase_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.spi_clock_polarity_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.SPI_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.spi_config_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.SPI_CTRLMASK (*C macro*), 665
drivers_RT_RT600_MIMXRT685S.SPI_DATA (*C macro*), 665
drivers_RT_RT600_MIMXRT685S.spi_data_width_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.SPI_DEASSERT_ALL (*C macro*), 665
drivers_RT_RT600_MIMXRT685S.SPI_DEASSERTNUM_SSEL (*C macro*), 665
drivers_RT_RT600_MIMXRT685S.spi_delay_config_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.spi_dma_callback_t (*C type*), 658
drivers_RT_RT600_MIMXRT685S.spi_dma_handle_t (*C type*), 657
drivers_RT_RT600_MIMXRT685S.SPI_DUMMYDATA (*C macro*), 665
drivers_RT_RT600_MIMXRT685S.SPI_FIFOTRIG_RXLVL_GET (*C macro*), 665
drivers_RT_RT600_MIMXRT685S.SPI_FIFOTRIG_TXLVL_GET (*C macro*), 665
drivers_RT_RT600_MIMXRT685S.SPI_FIFOWR_FLAGS_MASK (*C macro*), 665
drivers_RT_RT600_MIMXRT685S.spi_half_duplex_transfer_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.spi_master_callback_t (*C type*), 665
drivers_RT_RT600_MIMXRT685S.spi_master_config_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.spi_master_handle_t (*C type*), 665
drivers_RT_RT600_MIMXRT685S.SPI_MasterHalfDuplexTransferDMA (*C function*), 656
drivers_RT_RT600_MIMXRT685S.SPI_MasterTransferAbortDMA (*C function*), 657
drivers_RT_RT600_MIMXRT685S.SPI_MasterTransferCreateHandleDMA (*C function*), 655
drivers_RT_RT600_MIMXRT685S.SPI_MasterTransferDMA (*C function*), 655
drivers_RT_RT600_MIMXRT685S.SPI_MasterTransferGetCountDMA (*C function*), 657
drivers_RT_RT600_MIMXRT685S.SPI_RETRY_TIMES (*C macro*), 665
drivers_RT_RT600_MIMXRT685S.spi_rxfifo_watermark_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.spi_shift_direction_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.spi_slave_callback_t (*C type*), 665
drivers_RT_RT600_MIMXRT685S.spi_slave_config_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.spi_slave_handle_t (*C type*), 665
drivers_RT_RT600_MIMXRT685S.SPI_SlaveTransferAbortDMA (*C function*), 657
drivers_RT_RT600_MIMXRT685S.SPI_SlaveTransferCreateHandleDMA (*C function*), 656
drivers_RT_RT600_MIMXRT685S.SPI_SlaveTransferDMA (*C function*), 656
drivers_RT_RT600_MIMXRT685S.SPI_SlaveTransferGetCountDMA (*C function*), 657
drivers_RT_RT600_MIMXRT685S.spi_spol_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.spi_ssel_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.spi_transfer_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.spi_txfifo_watermark_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.spi_xfer_option_t (*C type*), 664
drivers_RT_RT600_MIMXRT685S.srcAddr (*C var*), 309
drivers_RT_RT600_MIMXRT685S.state (*C var*), 393
drivers_RT_RT600_MIMXRT685S.status_t (*C type*), 502
drivers_RT_RT600_MIMXRT685S.stereo_side_t (*C type*), 307
drivers_RT_RT600_MIMXRT685S.subaddressBuffer (*C var*), 393
drivers_RT_RT600_MIMXRT685S.subaddressCount (*C var*), 394
drivers_RT_RT600_MIMXRT685S.SUPPRESS_FALL_THROUGH_WARNING (*C macro*), 505
drivers_RT_RT600_MIMXRT685S.sys_pll_mult (*C var*), 262
drivers_RT_RT600_MIMXRT685S.sys_pll_mult_t (*C type*), 250
drivers_RT_RT600_MIMXRT685S.sys_pll_src (*C var*), 262
drivers_RT_RT600_MIMXRT685S.sys_pll_src_t (*C type*), 250
drivers_RT_RT600_MIMXRT685S.SYSCPUAHBCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.SYSCTL0_PDRCFGCLR_REG (*C macro*), 578

drivers_RT_RT600_MIMXRT685S.SYSCTL0_PDRCFGSET_REG (*C macro*), 578
drivers_RT_RT600_MIMXRT685S.SYSCTL0_TUPLE_REG (*C macro*), 578
drivers_RT_RT600_MIMXRT685S.SYSPLL0CLKSEL_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.SYSTICKFCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.SYSTICKFCLKSEL_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.transDataSize (*C var*), 394
drivers_RT_RT600_MIMXRT685S.transfer (*C var*), 394, 395
drivers_RT_RT600_MIMXRT685S.transferComplete (*C var*), 393
drivers_RT_RT600_MIMXRT685S.transferCount (*C var*), 393
drivers_RT_RT600_MIMXRT685S.trng_clock_mode_t (*C type*), 672
drivers_RT_RT600_MIMXRT685S.TRNG_CLOCKS (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.trng_config_t (*C type*), 672
drivers_RT_RT600_MIMXRT685S.TRNG_Deinit (*C function*), 673
drivers_RT_RT600_MIMXRT685S.TRNG_GetDefaultConfig (*C function*), 672
drivers_RT_RT600_MIMXRT685S.TRNG_GetRandomData (*C function*), 673
drivers_RT_RT600_MIMXRT685S.TRNG_Init (*C function*), 672
drivers_RT_RT600_MIMXRT685S.trng_ring_osc_div_t (*C type*), 672
drivers_RT_RT600_MIMXRT685S.TRNG_RSTS (*C macro*), 621
drivers_RT_RT600_MIMXRT685S.trng_sample_mode_t (*C type*), 672
drivers_RT_RT600_MIMXRT685S.trng_statistical_check_limit_t (*C type*), 672
drivers_RT_RT600_MIMXRT685S.txData (*C var*), 394
drivers_RT_RT600_MIMXRT685S.txDataSize (*C var*), 394
drivers_RT_RT600_MIMXRT685S.txDmaHandle (*C var*), 394, 395
drivers_RT_RT600_MIMXRT685S.UART_RETRY_TIMES (*C macro*), 692
drivers_RT_RT600_MIMXRT685S.UINT16_MAX (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.UINT32_MAX (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.UINT64_H (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.UINT64_L (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.UINTPTR_SIZE (*C macro*), 489
drivers_RT_RT600_MIMXRT685S.USART_ClearStatusFlags (*C function*), 680
drivers_RT_RT600_MIMXRT685S.usart_clock_polarity_t (*C type*), 692
drivers_RT_RT600_MIMXRT685S.USART_CLOCKS (*C macro*), 256
drivers_RT_RT600_MIMXRT685S.usart_config_t (*C type*), 692
drivers_RT_RT600_MIMXRT685S.usart_data_len_t (*C type*), 692
drivers_RT_RT600_MIMXRT685S.USART_Deinit (*C function*), 678
drivers_RT_RT600_MIMXRT685S.USART_DisableInterrupts (*C function*), 681
drivers_RT_RT600_MIMXRT685S.usart_dma_handle_t (*C type*), 677
drivers_RT_RT600_MIMXRT685S.usart_dma_transfer_callback_t (*C type*), 677
drivers_RT_RT600_MIMXRT685S.USART_Enable9bitMode (*C function*), 679
drivers_RT_RT600_MIMXRT685S.USART_Enable32kMode (*C function*), 679
drivers_RT_RT600_MIMXRT685S.USART_EnableAutoClearSCLK (*C function*), 681
drivers_RT_RT600_MIMXRT685S.USART_EnableContinuousSCLK (*C function*), 681
drivers_RT_RT600_MIMXRT685S.USART_EnableCTS (*C function*), 681
drivers_RT_RT600_MIMXRT685S.USART_EnableInterrupts (*C function*), 680
drivers_RT_RT600_MIMXRT685S.USART_EnableMatchAddress (*C function*), 680
drivers_RT_RT600_MIMXRT685S.USART_EnableRxDMA (*C function*), 681
drivers_RT_RT600_MIMXRT685S.USART_EnableTxDMA (*C function*), 681
drivers_RT_RT600_MIMXRT685S.USART_FIFOTRIG_RXLVL_GET (*C macro*), 692
drivers_RT_RT600_MIMXRT685S.USART_FIFOTRIG_TXLVL_GET (*C macro*), 692
drivers_RT_RT600_MIMXRT685S.USART_GetDefaultConfig (*C function*), 678
drivers_RT_RT600_MIMXRT685S.USART_GetEnabledInterrupts (*C function*), 681
drivers_RT_RT600_MIMXRT685S.USART_GetInstance (*C function*), 692
drivers_RT_RT600_MIMXRT685S.USART_GetRxFifoCount (*C function*), 682
drivers_RT_RT600_MIMXRT685S.USART_GetStatusFlags (*C function*), 680
drivers_RT_RT600_MIMXRT685S.USART_GetTxFifoCount (*C function*), 682
drivers_RT_RT600_MIMXRT685S.usart_handle_t (*C type*), 692
drivers_RT_RT600_MIMXRT685S.USART_Init (*C function*), 678
drivers_RT_RT600_MIMXRT685S.usart_parity_mode_t (*C type*), 691

drivers_RT_RT600_MIMXRT685S.USART_ReadBlocking (*C function*), 683
drivers_RT_RT600_MIMXRT685S.USART_ReadByte (*C function*), 682
drivers_RT_RT600_MIMXRT685S.usart_rxfifo_watermark_t (*C type*), 692
drivers_RT_RT600_MIMXRT685S.USART_SendAddress (*C function*), 682
drivers_RT_RT600_MIMXRT685S.USART_SetBaudRate (*C function*), 678
drivers_RT_RT600_MIMXRT685S.USART_SetMatchAddress (*C function*), 679
drivers_RT_RT600_MIMXRT685S.USART_SetRxFifoWatermark (*C function*), 682
drivers_RT_RT600_MIMXRT685S.USART_SetTxFifoWatermark (*C function*), 682
drivers_RT_RT600_MIMXRT685S.usart_stop_bit_count_t (*C type*), 691
drivers_RT_RT600_MIMXRT685S.usart_sync_mode_t (*C type*), 691
drivers_RT_RT600_MIMXRT685S.usart_transfer_callback_t (*C type*), 692
drivers_RT_RT600_MIMXRT685S.usart_transfer_t (*C type*), 692
drivers_RT_RT600_MIMXRT685S.USART_TransferAbortReceive (*C function*), 686
drivers_RT_RT600_MIMXRT685S.USART_TransferAbortReceiveDMA (*C function*), 676
drivers_RT_RT600_MIMXRT685S.USART_TransferAbortSend (*C function*), 685
drivers_RT_RT600_MIMXRT685S.USART_TransferAbortSendDMA (*C function*), 676
drivers_RT_RT600_MIMXRT685S.USART_TransferCreateHandle (*C function*), 683
drivers_RT_RT600_MIMXRT685S.USART_TransferCreateHandleDMA (*C function*), 675
drivers_RT_RT600_MIMXRT685S.USART_TransferGetReceiveCount (*C function*), 686
drivers_RT_RT600_MIMXRT685S.USART_TransferGetReceiveCountDMA (*C function*), 676
drivers_RT_RT600_MIMXRT685S.USART_TransferGetRxRingBufferLength (*C function*), 684
drivers_RT_RT600_MIMXRT685S.USART_TransferGetSendCount (*C function*), 685
drivers_RT_RT600_MIMXRT685S.USART_TransferGetSendCountDMA (*C function*), 676
drivers_RT_RT600_MIMXRT685S.USART_TransferHandleIRQ (*C function*), 686
drivers_RT_RT600_MIMXRT685S.USART_TransferReceiveDMA (*C function*), 675
drivers_RT_RT600_MIMXRT685S.USART_TransferReceiveNonBlocking (*C function*), 685
drivers_RT_RT600_MIMXRT685S.USART_TransferSendDMA (*C function*), 675
drivers_RT_RT600_MIMXRT685S.USART_TransferSendNonBlocking (*C function*), 683
drivers_RT_RT600_MIMXRT685S.USART_TransferStartRingBuffer (*C function*), 684
drivers_RT_RT600_MIMXRT685S.USART_TransferStopRingBuffer (*C function*), 684
drivers_RT_RT600_MIMXRT685S.usart_txfifo_watermark_t (*C type*), 692
drivers_RT_RT600_MIMXRT685S.USART_WriteBlocking (*C function*), 683
drivers_RT_RT600_MIMXRT685S.USART_WriteByte (*C function*), 682
drivers_RT_RT600_MIMXRT685S.USB_RSTS (*C macro*), 621
drivers_RT_RT600_MIMXRT685S.USBD_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.USBHSFCLKDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.USBHSFCLKSEL_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.USDHC_AdjustDelayForManualTuning (*C function*), 703
drivers_RT_RT600_MIMXRT685S.USDHC_ADMA1_ADDRESS_ALIGN (*C macro*), 722
drivers_RT_RT600_MIMXRT685S.USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK (*C macro*), 723
drivers_RT_RT600_MIMXRT685S.USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT (*C macro*), 722
drivers_RT_RT600_MIMXRT685S.USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK (*C macro*), 723
drivers_RT_RT600_MIMXRT685S.USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT (*C macro*), 723
drivers_RT_RT600_MIMXRT685S.USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (*C macro*), 723
drivers_RT_RT600_MIMXRT685S.usdhc_adma1_descriptor_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.USDHC_ADMA1_LENGTH_ALIGN (*C macro*), 722
drivers_RT_RT600_MIMXRT685S.USDHC_ADMA2_ADDRESS_ALIGN (*C macro*), 722
drivers_RT_RT600_MIMXRT685S.USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK (*C macro*), 724
drivers_RT_RT600_MIMXRT685S.USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT (*C macro*), 723
drivers_RT_RT600_MIMXRT685S.USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (*C macro*), 724

drivers_RT_RT600_MIMXRT685S.usdhc_adma2_descriptor_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.USDHC_ADMA2_LENGTH_ALIGN (*C macro*), 722
drivers_RT_RT600_MIMXRT685S.usdhc_adma_config_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.USDHC_AssertHardwareReset (*C function*), 699
drivers_RT_RT600_MIMXRT685S.usdhc_boot_config_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.usdhc_boot_mode_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.usdhc_burst_len_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.usdhc_capability_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.usdhc_card_command_type_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.usdhc_card_response_type_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.USDHC_CardDetectByData3 (*C function*), 700
drivers_RT_RT600_MIMXRT685S.USDHC_CheckStdTuningResult (*C function*), 703
drivers_RT_RT600_MIMXRT685S.USDHC_CheckTuningError (*C function*), 704
drivers_RT_RT600_MIMXRT685S.USDHC_ClearInterruptStatusFlags (*C function*), 698
drivers_RT_RT600_MIMXRT685S.USDHC_CLOCKS (*C macro*), 258
drivers_RT_RT600_MIMXRT685S.usdhc_command_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.usdhc_config_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.USDHC_ConfigStrobeDLL (*C function*), 704
drivers_RT_RT600_MIMXRT685S.usdhc_data_bus_width_t (*C type*), 720
drivers_RT_RT600_MIMXRT685S.usdhc_data_t (*C type*), 722
drivers_RT_RT600_MIMXRT685S.USDHC_Deinit (*C function*), 696
drivers_RT_RT600_MIMXRT685S.USDHC_DetectCardInsert (*C function*), 700
drivers_RT_RT600_MIMXRT685S.USDHC_DisableInterruptSignal (*C function*), 698
drivers_RT_RT600_MIMXRT685S.USDHC_DisableInterruptStatus (*C function*), 698
drivers_RT_RT600_MIMXRT685S.usdhc_dma_mode_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.USDHC_DriverIRQHandler (*C function*), 706
drivers_RT_RT600_MIMXRT685S.USDHC_EnableAutoTuning (*C function*), 702
drivers_RT_RT600_MIMXRT685S.USDHC_EnableAutoTuningForCmdAndData (*C function*), 702
drivers_RT_RT600_MIMXRT685S.USDHC_EnableDDRMode (*C function*), 704
drivers_RT_RT600_MIMXRT685S.USDHC_EnableHS400Mode (*C function*), 704
drivers_RT_RT600_MIMXRT685S.USDHC_EnableInternalDMA (*C function*), 697
drivers_RT_RT600_MIMXRT685S.USDHC_EnableInterruptSignal (*C function*), 698
drivers_RT_RT600_MIMXRT685S.USDHC_EnableInterruptStatus (*C function*), 697
drivers_RT_RT600_MIMXRT685S.USDHC_EnableManualTuning (*C function*), 702
drivers_RT_RT600_MIMXRT685S.USDHC_EnableMmcBoot (*C function*), 701
drivers_RT_RT600_MIMXRT685S.USDHC_EnableSdioControl (*C function*), 701
drivers_RT_RT600_MIMXRT685S.USDHC_EnableStandardTuning (*C function*), 703
drivers_RT_RT600_MIMXRT685S.USDHC_EnableStrobeDLL (*C function*), 704
drivers_RT_RT600_MIMXRT685S.USDHC_EnableWakeupEvent (*C function*), 700
drivers_RT_RT600_MIMXRT685S.usdhc_endian_mode_t (*C type*), 720
drivers_RT_RT600_MIMXRT685S.USDHC_ForceClockOn (*C function*), 699
drivers_RT_RT600_MIMXRT685S.USDHC_GetAdmaErrorStatusFlags (*C function*), 698
drivers_RT_RT600_MIMXRT685S.USDHC_GetAutoCommand12ErrorStatusFlags (*C function*), 698
drivers_RT_RT600_MIMXRT685S.USDHC_GetCapability (*C function*), 699
drivers_RT_RT600_MIMXRT685S.USDHC_GetEnabledInterruptStatusFlags (*C function*), 698
drivers_RT_RT600_MIMXRT685S.USDHC_GetExecuteStdTuningStatus (*C function*), 703
drivers_RT_RT600_MIMXRT685S.USDHC_GetInterruptStatusFlags (*C function*), 698
drivers_RT_RT600_MIMXRT685S.USDHC_GetPresentStatusFlags (*C function*), 699
drivers_RT_RT600_MIMXRT685S.USDHC_GetStrobeDLLStatus (*C function*), 704
drivers_RT_RT600_MIMXRT685S.USDHC_GetTuningDelayStatus (*C function*), 702
drivers_RT_RT600_MIMXRT685S.usdhc_handle_t (*C type*), 722
drivers_RT_RT600_MIMXRT685S.usdhc_host_t (*C type*), 722
drivers_RT_RT600_MIMXRT685S.USDHC_Init (*C function*), 695
drivers_RT_RT600_MIMXRT685S.USDHC_MAX_BLOCK_COUNT (*C macro*), 722
drivers_RT_RT600_MIMXRT685S.USDHC_ReadData (*C function*), 700
drivers_RT_RT600_MIMXRT685S.USDHC_RequestReTuning (*C function*), 702
drivers_RT_RT600_MIMXRT685S.USDHC_RequestStopAtBlockGap (*C function*), 701
drivers_RT_RT600_MIMXRT685S.USDHC_RequestTuningForSDR50 (*C function*), 702

drivers_RT_RT600_MIMXRT685S.USDHC_Reset (*C function*), 696
drivers_RT_RT600_MIMXRT685S.USDHC_ResetStrobeDLL (*C function*), 704
drivers_RT_RT600_MIMXRT685S.USDHC_RSTS (*C macro*), 621
drivers_RT_RT600_MIMXRT685S.usdhc_scatter_gather_data_list_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.usdhc_scatter_gather_data_t (*C type*), 721
drivers_RT_RT600_MIMXRT685S.usdhc_scatter_gather_transfer_t (*C type*), 722
drivers_RT_RT600_MIMXRT685S.USDHC_SendCommand (*C function*), 700
drivers_RT_RT600_MIMXRT685S.USDHC_SetADMA1Descriptor (*C function*), 697
drivers_RT_RT600_MIMXRT685S.USDHC_SetADMA2Descriptor (*C function*), 697
drivers_RT_RT600_MIMXRT685S.USDHC_SetAdmaTableConfig (*C function*), 696
drivers_RT_RT600_MIMXRT685S.USDHC_SetCardActive (*C function*), 699
drivers_RT_RT600_MIMXRT685S.USDHC_SetContinueRequest (*C function*), 701
drivers_RT_RT600_MIMXRT685S.USDHC_SetDataBusWidth (*C function*), 700
drivers_RT_RT600_MIMXRT685S.USDHC_SetDataConfig (*C function*), 705
drivers_RT_RT600_MIMXRT685S.USDHC_SetForceEvent (*C function*), 701
drivers_RT_RT600_MIMXRT685S.USDHC_SetInternalDmaConfig (*C function*), 696
drivers_RT_RT600_MIMXRT685S.USDHC_SetMmcBootConfig (*C function*), 701
drivers_RT_RT600_MIMXRT685S.USDHC_SetSdClock (*C function*), 699
drivers_RT_RT600_MIMXRT685S.USDHC_SetStandardTuningCounter (*C function*), 703
drivers_RT_RT600_MIMXRT685S.USDHC_SetStrobeDllOverride (*C function*), 704
drivers_RT_RT600_MIMXRT685S.USDHC_SetTuningDelay (*C function*), 702
drivers_RT_RT600_MIMXRT685S.usdhc_transfer_callback_t (*C type*), 722
drivers_RT_RT600_MIMXRT685S.usdhc_transfer_direction_t (*C type*), 720
drivers_RT_RT600_MIMXRT685S.usdhc_transfer_function_t (*C type*), 722
drivers_RT_RT600_MIMXRT685S.usdhc_transfer_t (*C type*), 722
drivers_RT_RT600_MIMXRT685S.USDHC_TransferBlocking (*C function*), 705
drivers_RT_RT600_MIMXRT685S.USDHC_TransferCreateHandle (*C function*), 705
drivers_RT_RT600_MIMXRT685S.USDHC_TransferHandleIRQ (*C function*), 706
drivers_RT_RT600_MIMXRT685S.USDHC_TransferNonBlocking (*C function*), 705
drivers_RT_RT600_MIMXRT685S.USDHC_WriteData (*C function*), 700
drivers_RT_RT600_MIMXRT685S.USEC_TO_COUNT (*C macro*), 490
drivers_RT_RT600_MIMXRT685S.userData (*C var*), 394, 395
drivers_RT_RT600_MIMXRT685S.utick_callback_t (*C type*), 731
drivers_RT_RT600_MIMXRT685S.UTICK_ClearStatusFlags (*C function*), 730
drivers_RT_RT600_MIMXRT685S.UTICK_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.UTICK_Deinit (*C function*), 730
drivers_RT_RT600_MIMXRT685S.UTICK_GetStatusFlags (*C function*), 730
drivers_RT_RT600_MIMXRT685S.UTICK_HandleIRQ (*C function*), 730
drivers_RT_RT600_MIMXRT685S.UTICK_Init (*C function*), 730
drivers_RT_RT600_MIMXRT685S.utick_mode_t (*C type*), 731
drivers_RT_RT600_MIMXRT685S.UTICK_RSTS (*C macro*), 621
drivers_RT_RT600_MIMXRT685S.UTICK_SetTick (*C function*), 730
drivers_RT_RT600_MIMXRT685S.UTICKFCLKSEL_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.Vdde0Range (*C var*), 578
drivers_RT_RT600_MIMXRT685S.Vdde1Range (*C var*), 578
drivers_RT_RT600_MIMXRT685S.Vdde2Range (*C var*), 578
drivers_RT_RT600_MIMXRT685S.WAKECLK32KHZDIV_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.WAKECLK32KHZSEL_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.wasTransmit (*C var*), 395
drivers_RT_RT600_MIMXRT685S.WDT0FCLKSEL_OFFSET (*C macro*), 259
drivers_RT_RT600_MIMXRT685S.WDT1FCLKSEL_OFFSET (*C macro*), 261
drivers_RT_RT600_MIMXRT685S.workaroundBuff (*C var*), 394
drivers_RT_RT600_MIMXRT685S.WWDT_ClearStatusFlags (*C function*), 732
drivers_RT_RT600_MIMXRT685S.WWDT_CLOCKS (*C macro*), 257
drivers_RT_RT600_MIMXRT685S.wwdt_config_t (*C type*), 734
drivers_RT_RT600_MIMXRT685S.WWDT_Deinit (*C function*), 732
drivers_RT_RT600_MIMXRT685S.WWDT_Disable (*C function*), 732
drivers_RT_RT600_MIMXRT685S.WWDT_Enable (*C function*), 732

drivers_RT_RT600_MIMXRT685S.WWDT_FIRST_WORD_OF_REFRESH (*C macro*), 734
drivers_RT_RT600_MIMXRT685S.WWDT_GetDefaultConfig (*C function*), 731
drivers_RT_RT600_MIMXRT685S.WWDT_GetStatusFlags (*C function*), 732
drivers_RT_RT600_MIMXRT685S.WWDT_Init (*C function*), 731
drivers_RT_RT600_MIMXRT685S.WWDT_Refresh (*C function*), 733
drivers_RT_RT600_MIMXRT685S.WWDT_RSTS (*C macro*), 621
drivers_RT_RT600_MIMXRT685S.WWDT_SECOND_WORD_OF_REFRESH (*C macro*), 734
drivers_RT_RT600_MIMXRT685S.WWDT_SetTimeoutValue (*C function*), 733
drivers_RT_RT600_MIMXRT685S.WWDT_SetWarningValue (*C function*), 733
drivers_RT_RT600_MIMXRT685S.WWDT_SetWindowValue (*C function*), 733