MCUXpresso SDK Documentation

Release 26.09.00

NXP
Sep 29, 2026

Table of contents

1 FRDM-K22F **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 MCUXpresso SDK Package . . . 7
- 1.4 Getting Started with MCUXpresso SDK GitHub . . . 63
  - 1.4.1 Getting Started with MCUXpresso SDK Repository . . . 63
- 1.5 Release Notes . . . 75
  - 1.5.1 MCUXpresso SDK Release Notes . . . 75
- 1.6 ChangeLog . . . 79
  - 1.6.1 MCUXpresso SDK Changelog . . . 79
- 1.7 Driver API Reference Manual . . . 136
- 1.8 Middleware Documentation . . . 137
  - 1.8.1 Multicore . . . 137
  - 1.8.2 FreeMASTER . . . 137
  - 1.8.3 FreeRTOS . . . 137
  - 1.8.4 File systemFatfs . . . 137

2 MK22F51212 **139**
- 2.1 ADC16: 16-bit SAR Analog-to-Digital Converter Driver . . . 139
- 2.2 Clock Driver . . . 149
- 2.3 CMP: Analog Comparator Driver . . . 177
- 2.4 CRC: Cyclic Redundancy Check Driver . . . 182
- 2.5 DAC: Digital-to-Analog Converter Driver . . . 185
- 2.6 DMAMUX: Direct Memory Access Multiplexer Driver . . . 190
- 2.7 DSPI: Serial Peripheral Interface Driver . . . 192
- 2.8 DSPI Driver . . . 192
- 2.9 DSPI eDMA Driver . . . 219
- 2.10 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver . . . 225
- 2.11 EWM: External Watchdog Monitor Driver . . . 247
- 2.12 FGPIO Driver . . . 250
- 2.13 C90TFS Flash Driver . . . 250
- 2.14 FlexBus: External Bus Interface Driver . . . 250
- 2.15 ftfx adapter . . . 257
- 2.16 Ftftx CACHE Driver . . . 257
- 2.17 ftfx controller . . . 259
- 2.18 ftfx feature . . . 277
- 2.19 Ftftx FLASH Driver . . . 277
- 2.20 Ftftx FLEXNVM Driver . . . 292
- 2.21 ftfx utilities . . . 303
- 2.22 FTM: FlexTimer Driver . . . 304
- 2.23 GPIO: General-Purpose Input/Output Driver . . . 330
- 2.24 GPIO Driver . . . 331
- 2.25 I2C: Inter-Integrated Circuit Driver . . . 333
- 2.26 I2C Driver . . . 333

2.27 I2C eDMA Driver . . . 349
2.28 Common Driver . . . 351
2.29 Lin_lpuart_driver . . . 367
2.30 LLWU: Low-Leakage Wakeup Unit Driver . . . 376
2.31 LPTMR: Low-Power Timer . . . 380
2.32 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver . . . 387
2.33 LPUART Driver . . . 387
2.34 LPUART eDMA Driver . . . 408
2.35 MCM: Miscellaneous Control Module . . . 411
2.36 PDB: Programmable Delay Block . . . 417
2.37 PIT: Periodic Interrupt Timer . . . 429
2.38 PMC: Power Management Controller . . . 433
2.39 PORT: Port Control and Interrupts . . . 439
2.40 RCM: Reset Control Module Driver . . . 448
2.41 RNGA: Random Number Generator Accelerator Driver . . . 454
2.42 RTC: Real Time Clock . . . 456
2.43 SAI: Serial Audio Interface . . . 465
2.44 SAI Driver . . . 465
2.45 SAI EDMA Driver . . . 494
2.46 SIM: System Integration Module Driver . . . 501
2.47 Smart Card . . . 503
2.48 Smart Card PHY Driver . . . 513
2.49 Smart Card PHY GPIO Driver . . . 515
2.50 Smart Card PHY TDA8035 Driver . . . 515
2.51 Smart Card UART Driver . . . 515
2.52 SMC: System Mode Controller Driver . . . 518
2.53 UART: Universal Asynchronous Receiver/Transmitter Driver . . . 525
2.54 UART Driver . . . 525
2.55 UART eDMA Driver . . . 542
2.56 VREF: Voltage Reference Driver . . . 545
2.57 WDOG: Watchdog Timer Driver . . . 549

3 Middleware **557**
3.1 File System . . . 557
3.1.1 FatFs . . . 557
3.2 Motor Control . . . 559
3.2.1 FreeMASTER . . . 559
3.3 Multicore SDK . . . 597
3.3.1 Quick links . . . 597
3.3.2 Components . . . 598

4 RTOS **689**
4.1 FreeRTOS . . . 689
4.1.1 FreeRTOS kernel . . . 689
4.1.2 FreeRTOS drivers . . . 693
4.1.3 backoffalgorithm . . . 693
4.1.4 corehttp . . . 693
4.1.5 corejson . . . 693
4.1.6 coremqtt . . . 693
4.1.7 corepkcs11 . . . 694
4.1.8 freertos-plus-tcp . . . 694

5 Downloads **695**
5.1 boards . . . 695
5.2 develop . . . 695

Index **697**

This documentation contains information specific to the frdmk22f board.

Chapter 1

FRDM-K22F

1.1 Overview

Form-factor compatible with the Arduino R3 pin layout.Peripherals enable rapid prototyping, including a 6-axis digital accelerometer and magnetometer to create full eCompass capabilities, a tri-colored LED and 2 user push-buttons for direct interaction, and connectivity for use with Bluetooth and 2.4 GHz radio add-on modules.OpenSDAv2 offers options for serial communication, flash programming, and run-control debugging.The FRDM-K22F is enabled with the ARM mbed development platform, which means you can quickly get started designing with no downloads or installations.

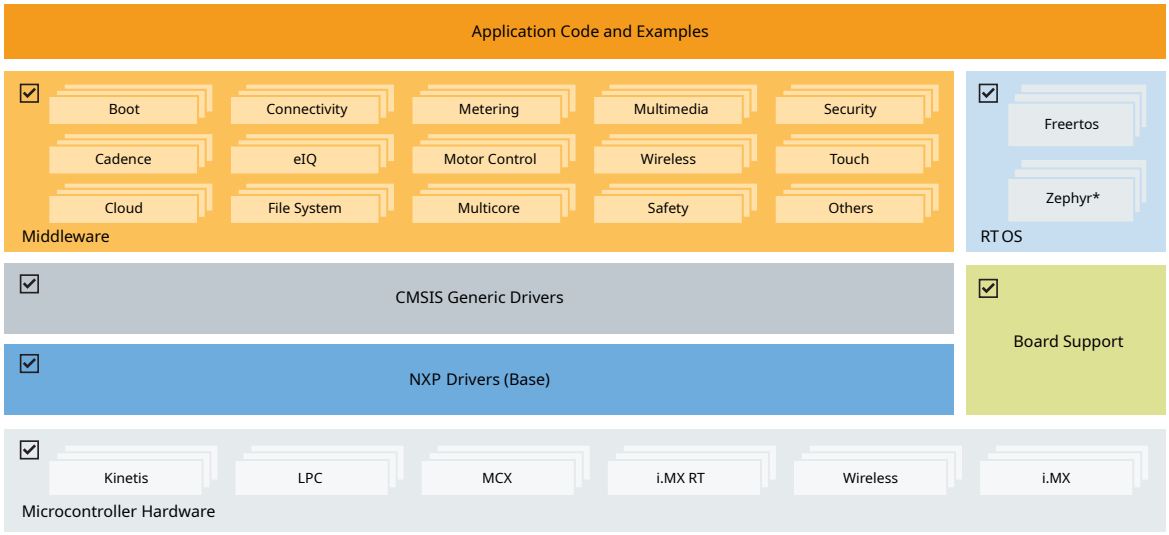
MCU device and part on board is shown below:

- Device: MK22F51212
- PartNumber: MK22FN512VLH12

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 RTOS 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 MCUXpresso SDK Package

Starting with version 25.09.00, MCUXpresso SDK introduced two package versions for offline development:

- **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 SDK development experience. Available when selecting the ARMGCC toolchain.

From version 25.12.00 onward:

- When you select ARMGCC, the SDK download will use the Repository-Layout version.
- For all other toolchains, the SDK download will remain in the Classic version.

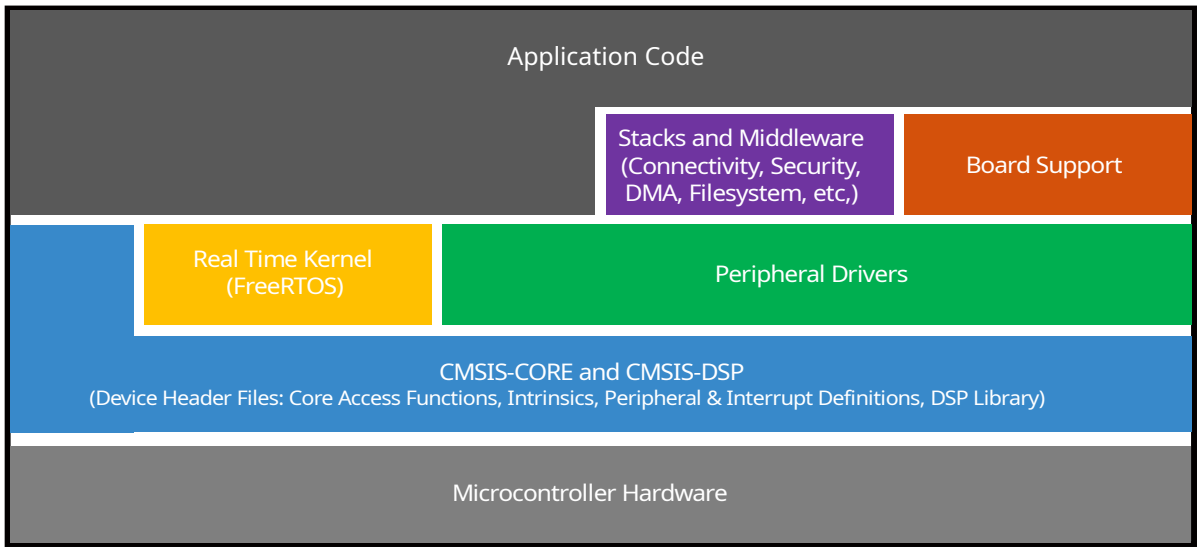
Note: The Repository-Layout SDK package was first introduced in version 25.09.00, but initially only for MCXW23x platforms.

Classic SDK Package

Overview The NXP MCUXpresso software and tools offer comprehensive development solutions designed to optimize, ease, and help accelerate embedded system development of applications based on general purpose, crossover, and Bluetooth-enabled MCUs from NXP. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications. Along with the peripheral drivers, the MCUXpresso SDK provides an extensive and rich set of example applications covering everything from basic peripheral use case examples to full demo applications. The MCUXpresso SDK contains optional RTOS integrations such as FreeRTOS and Azure RTOS, and various other middleware to support rapid development.

For supported toolchain versions, see *MCUXpresso SDK Release Notes* (document MCUXSDKRN).

For more details about MCUXpresso SDK, see MCUXpresso Software Development Kit (SDK).



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

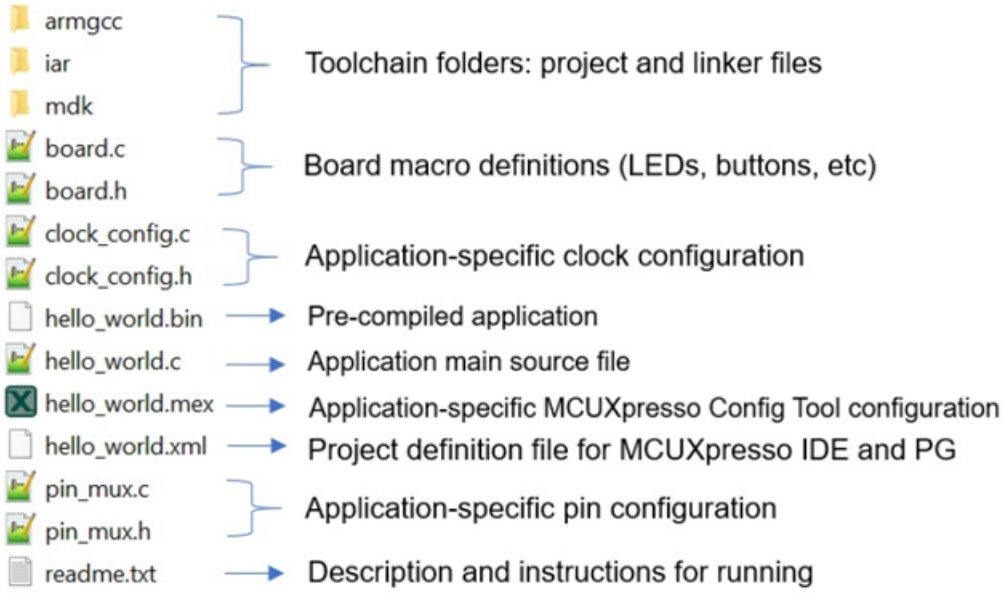
- cmsis_driver_examples: Simple applications intended to show how to use CMSIS drivers.
- demo_apps: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- driver_examples: Simple applications that show how to use the MCUXpresso SDK'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).
- emwin_examples: Applications that use the emWin GUI widgets.
- rtos_examples: Basic FreeRTOS OS examples that show the use of various RTOS objects (semaphores, queues, and so on) and interfaces with the MCUXpresso SDK's RTOS drivers
- usb_examples: Applications that use the USB host/device/OTG stack.

Example application structure

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

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

In the hello_world application folder you see the following contents:



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

Locating example application source files

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

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

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

For examples containing middleware/stacks or an RTOS, there are references to the appropriate source code. Middleware source files are located in the middleware folder and 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.

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 hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

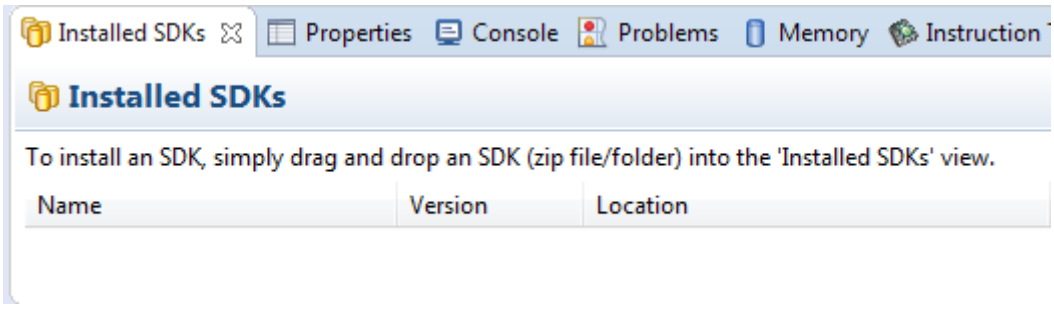
Select the workspace location

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.

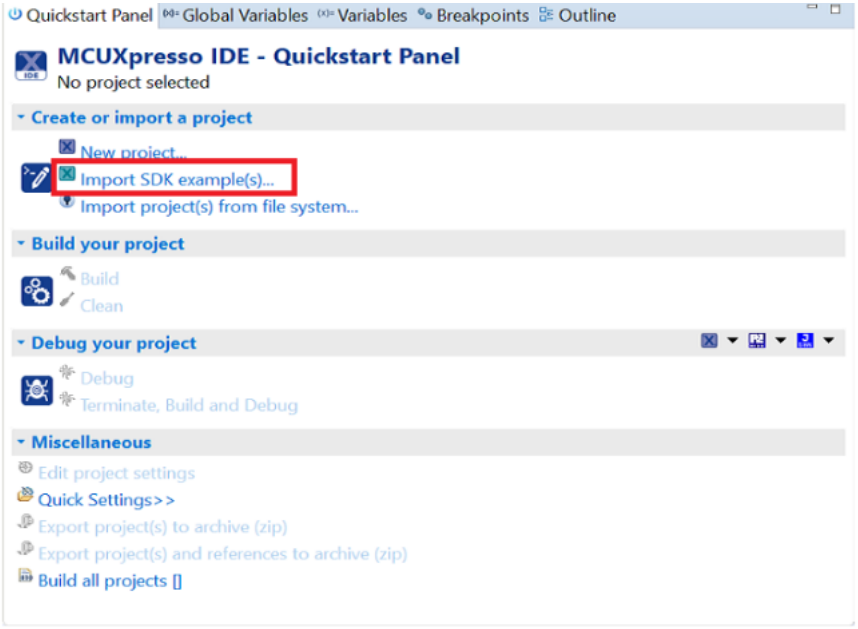
Build an example application

To build an example application, follow these steps.

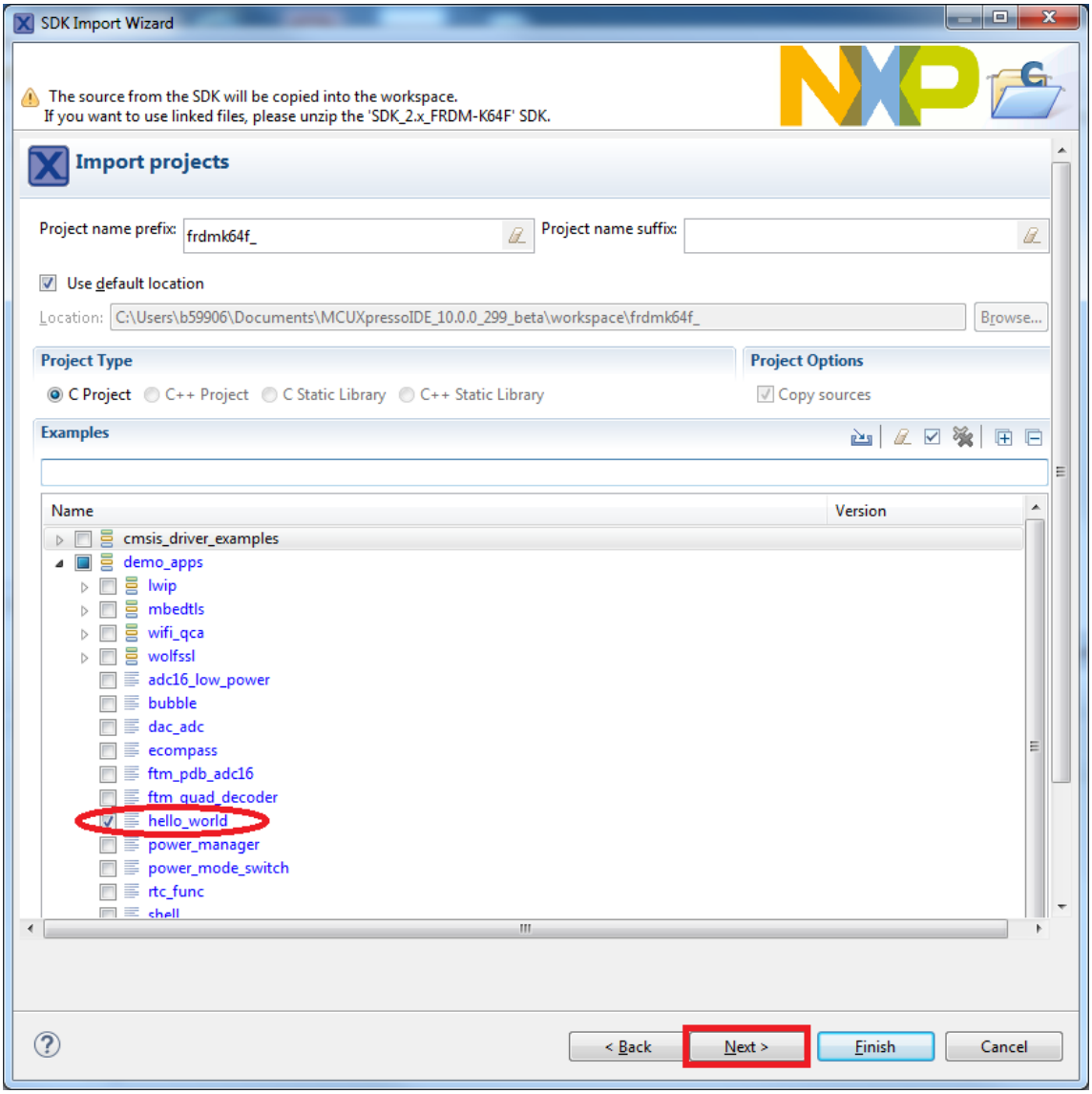
1. Drag and drop the SDK zip file into the **Installed SDKs** view to install an SDK. In the window that appears, click **OK** and wait until the import has finished.



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



3. Expand the demo_apps folder and select hello_world.
4. Click **Next.**



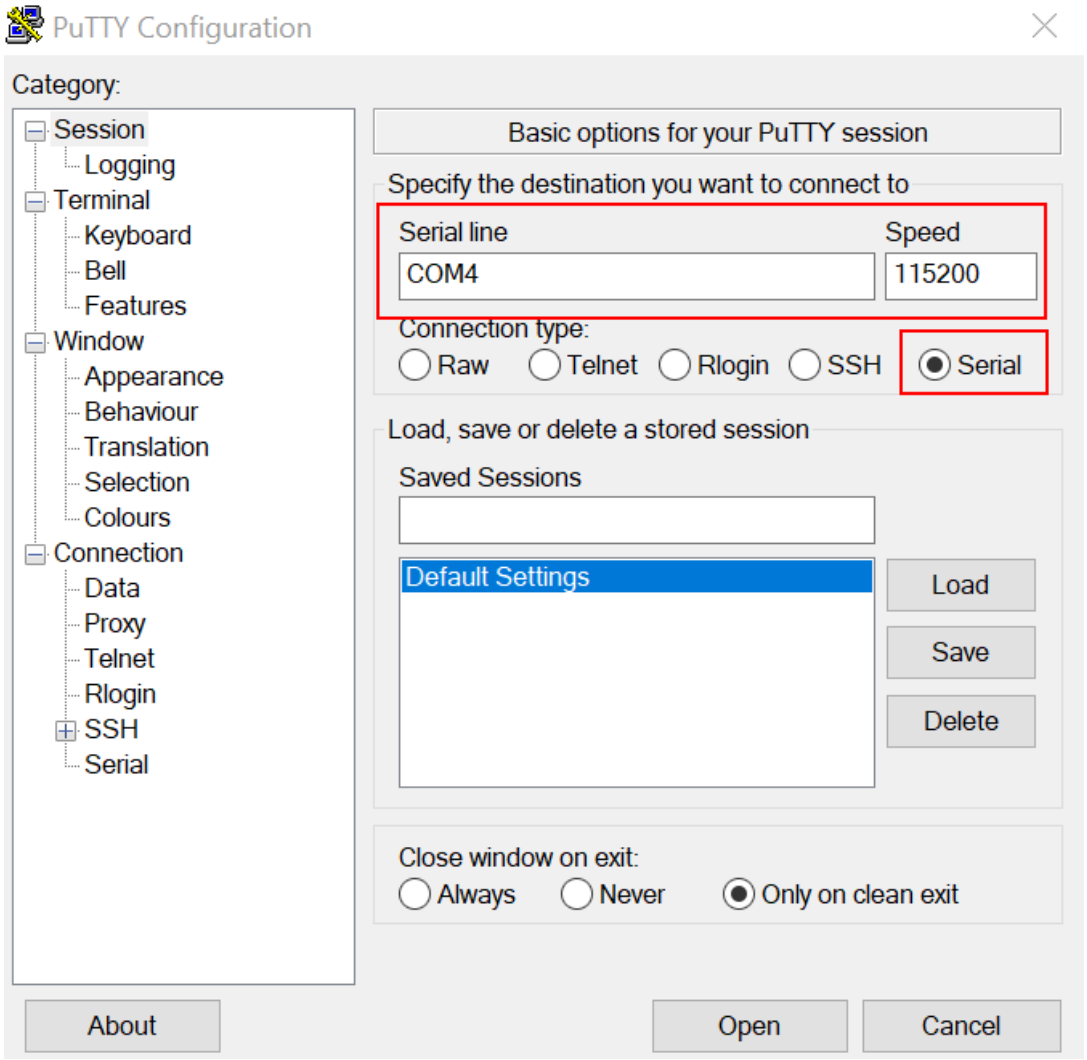
5. Ensure **Redlib: Use floating-point version of printf** is selected if the example prints floating-point numbers on the terminalfor demo applications such as adc_basic, adc_burst, adc_dma, and adc_interrupt. Otherwise, it is not necessary to select this option. Then, click **Finish**.

Run an example application

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

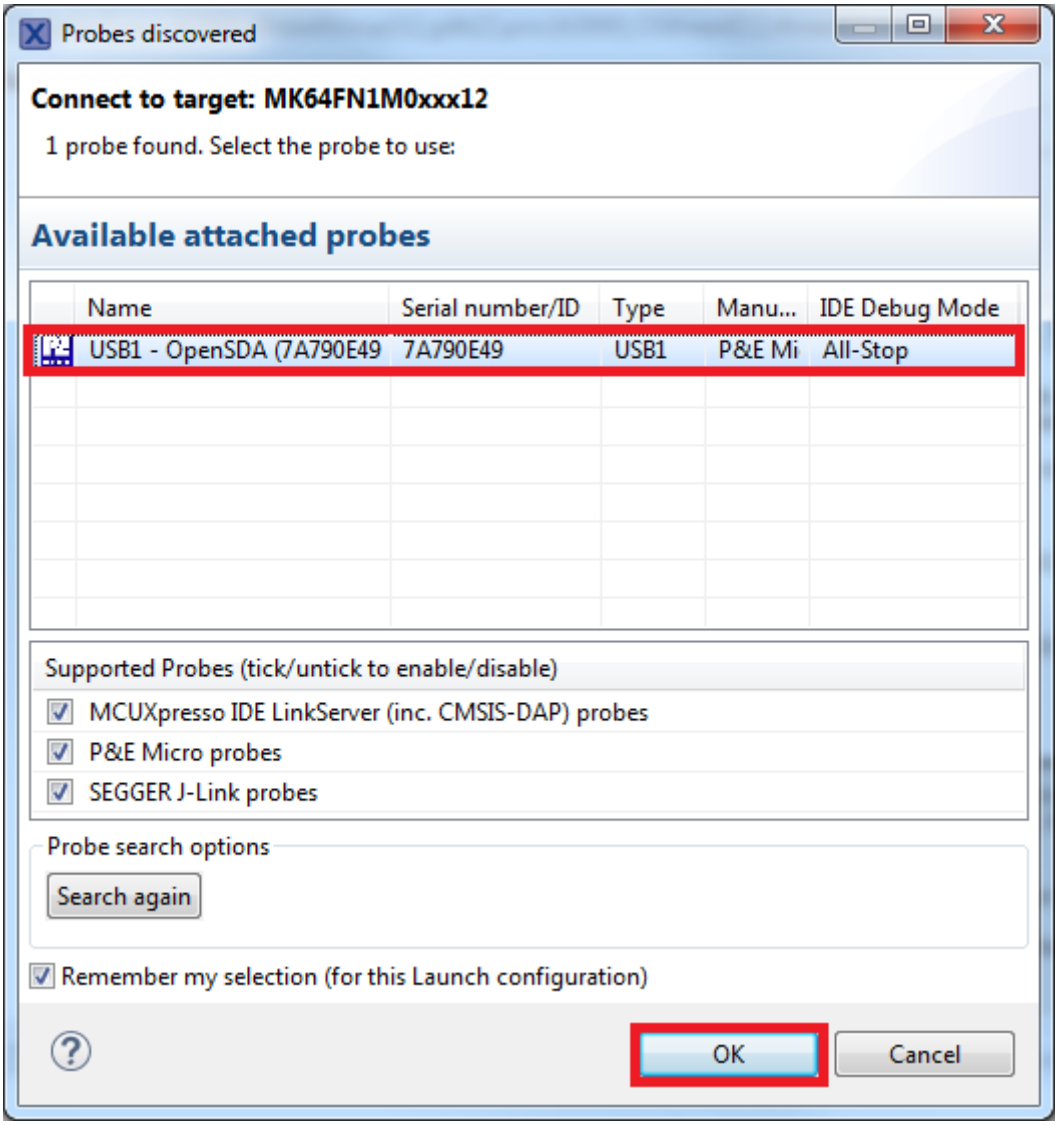
To download and run the application, perform the following steps:

1. Ensure the host driver for the debugger firmware has been installed.
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



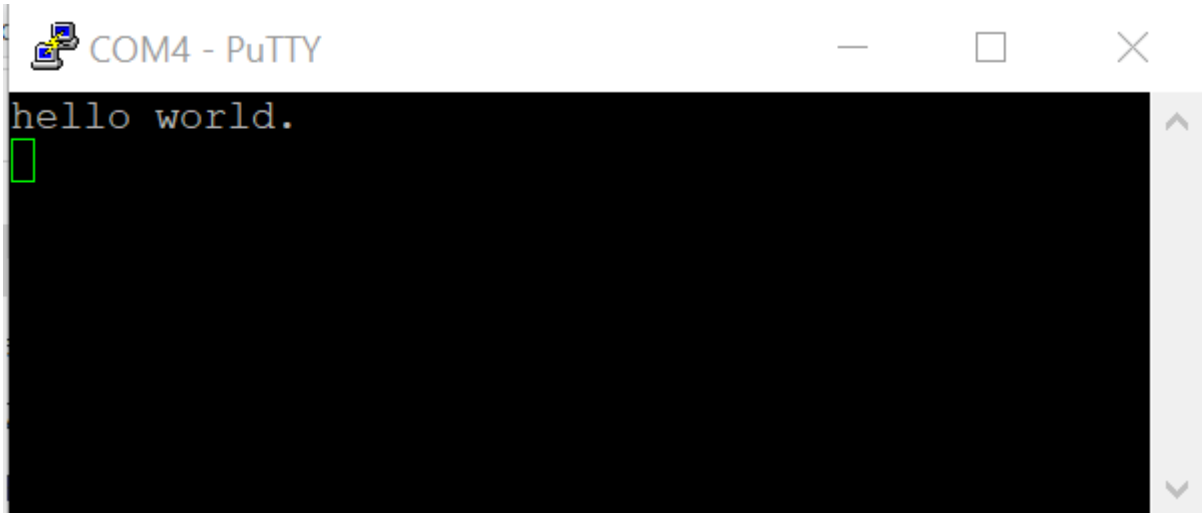
4. 1 stop bit

4. On the **Quickstart Panel**, click **Debug** 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.)



6. The application is downloaded to the target and automatically runs to main().
7. Start the application by clicking **Resume**.

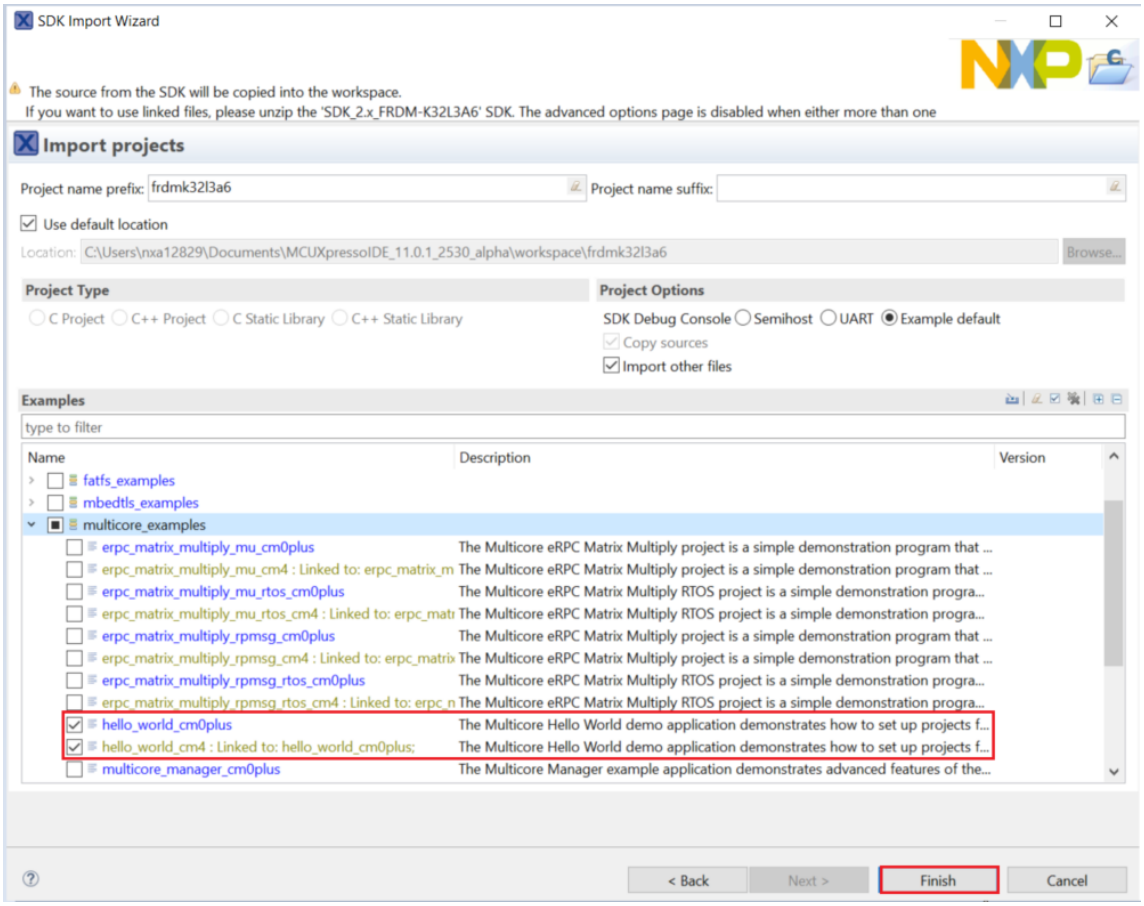
The hello_world application is now running and a banner is displayed on the terminal. If not, check your terminal settings and connections.



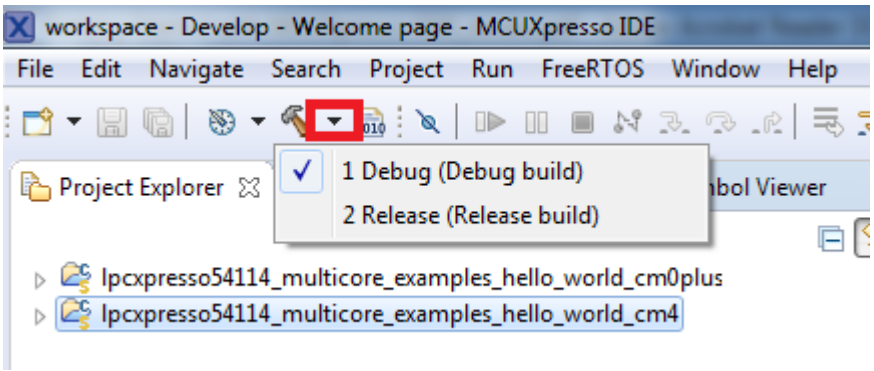
Build a multicore example application

This section describes the steps required to configure MCUXpresso IDE to build, run, and debug multicore example applications. The following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello_world example application targeted for the LPCXpresso54114 hardware platform is used as an example.

1. Multicore examples are imported into the workspace in a similar way as single core applications, explained in **Build an example application**. When the SDK zip package for LPCXpresso54114 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 **LPCxx** folder and select **LPC54114J256**. Then, select **lpcxpresso54114** and click **Next**.
2. Expand the multicore_examples/hello_world folder and select **cm4**. The cm0plus counterpart project is automatically imported with the cm4 project, because the multicore examples are linked together and there is no need to select it explicitly. Click **Finish**.

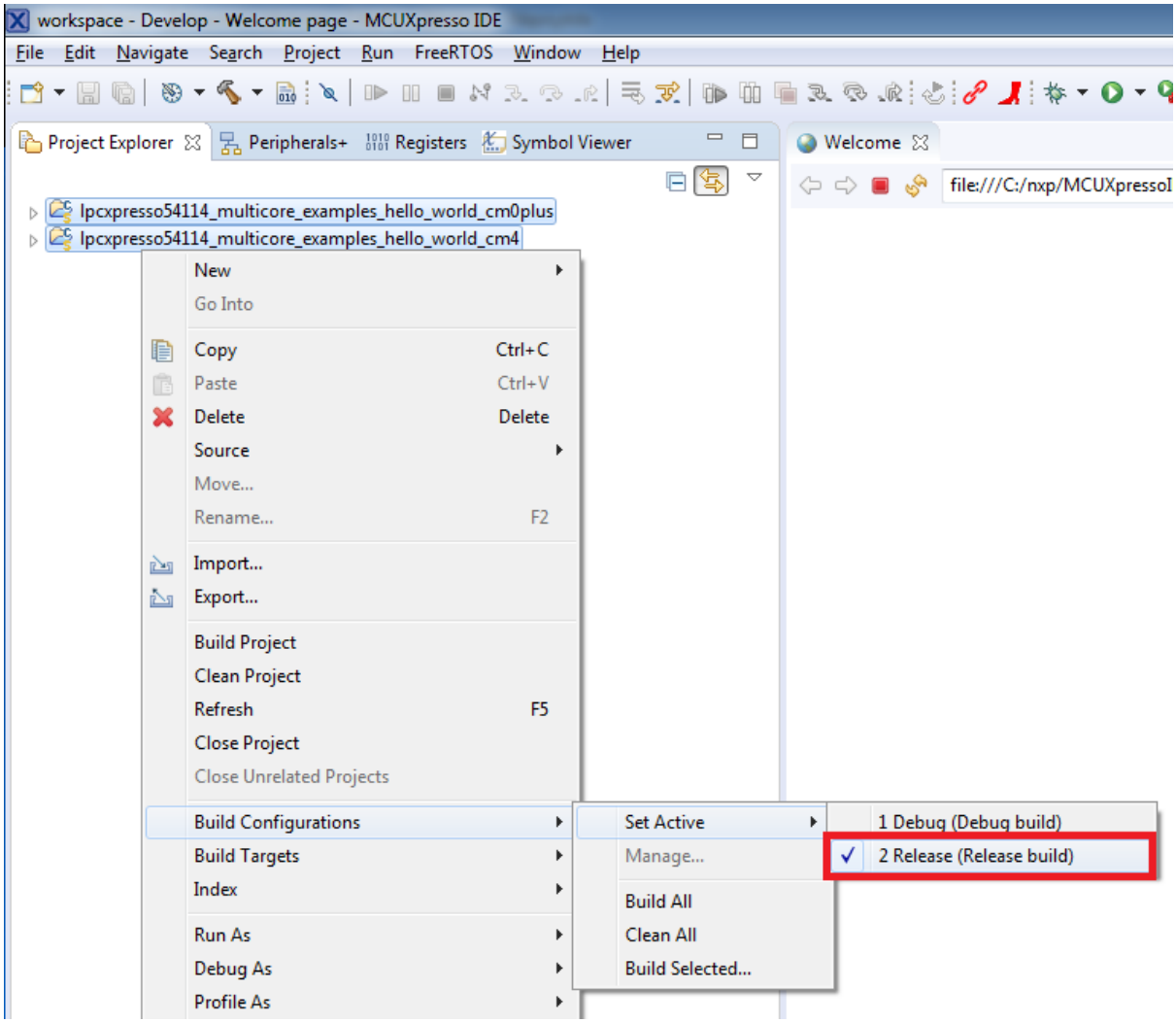


3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the lpcxpresso54114_multicore_examples_hello_world_cm4 project (multicore master project) in the Project Explorer. Then choose the appropriate build target, **Debug** or **Release**, by clicking the downward facing arrow next to the hammer icon, as shown in the figure. For this example, select **Debug**.



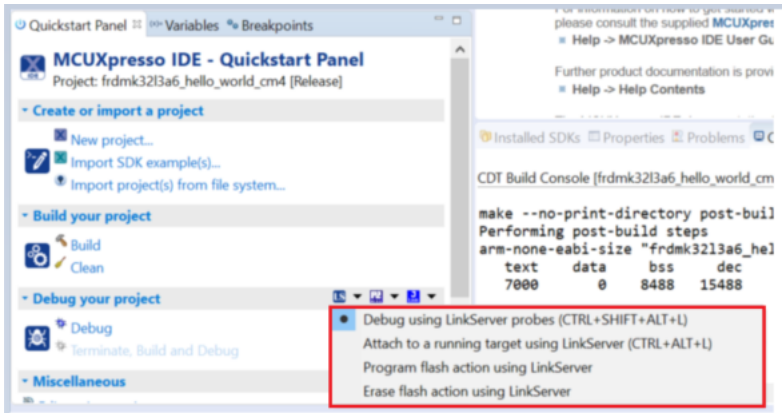
The project starts building after the build target is selected. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm4) also causes the referenced auxiliary core application (cm0plus) to build.

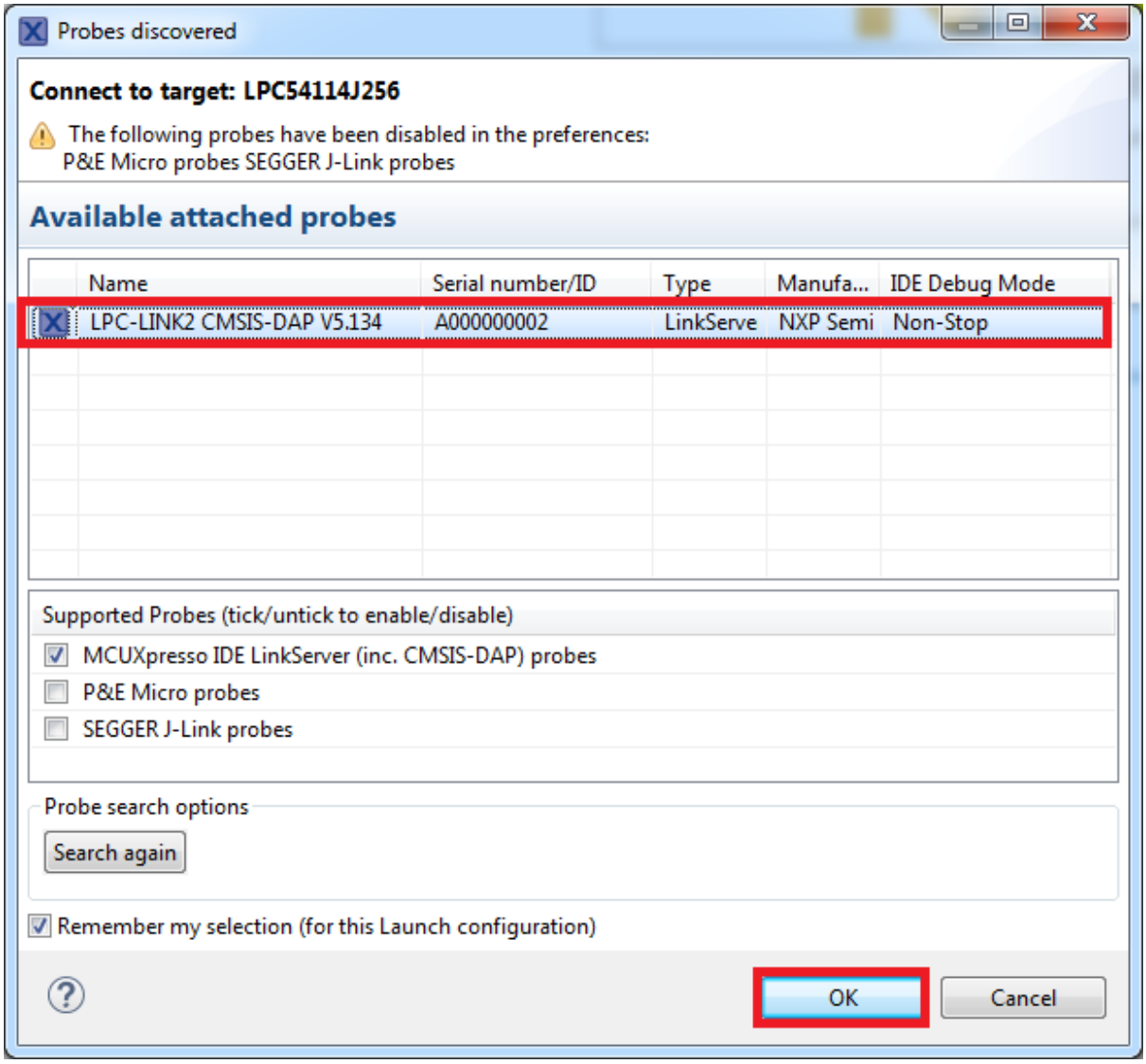
Note: When the **Release** build is requested, it is necessary to change the build configuration of both the primary and auxiliary core application projects first. To do this, select both projects in the Project Explorer view and then right click which displays the context-sensitive menu. Select **Build Configurations** -> **Set Active** -> **Release**. This alternate navigation using the menu item is **Project** -> **Build Configuration** -> **Set Active** -> **Release**. After switching to the **Release** build configuration, the build of the multicore example can be started by triggering the primary core application (cm4) build.

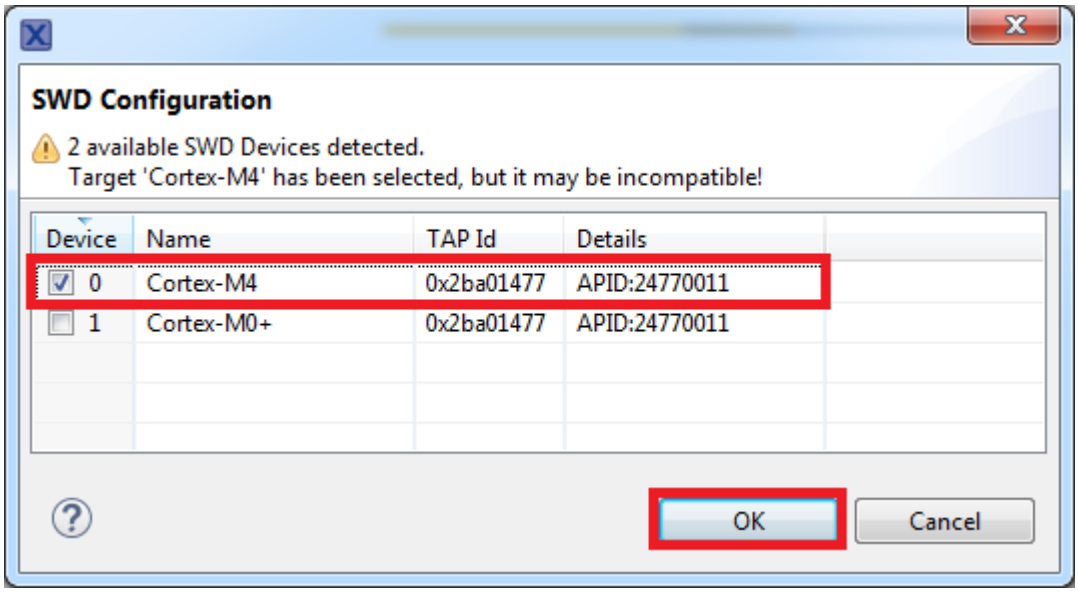


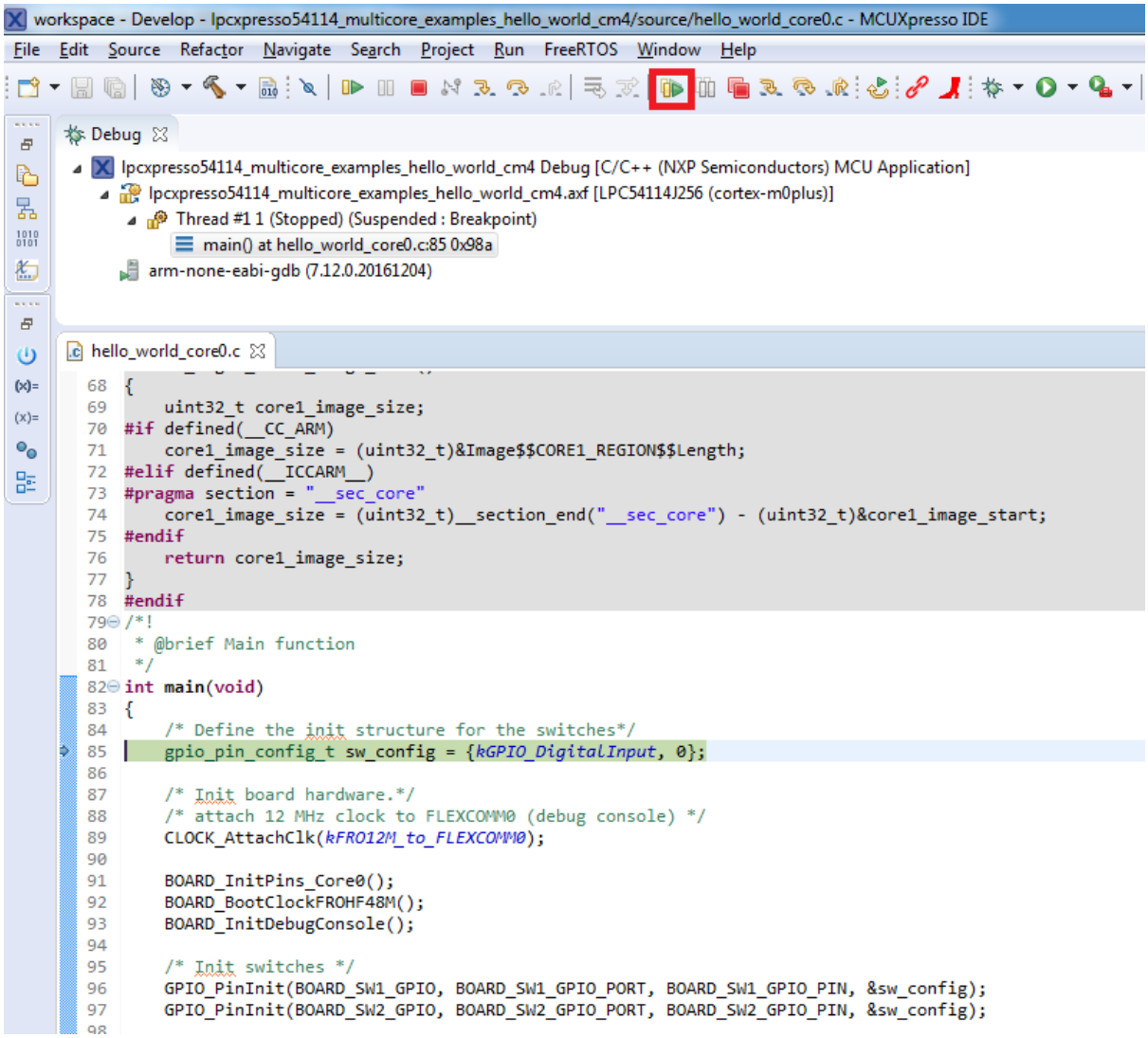
Run a multicore example application

The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in **Run an example application**. These steps are common for both single-core applications and the primary side of dual-core applications, ensuring both sides of the multicore application are properly loaded and started. However, there is one additional dialogue that is specific to multicore examples which requires selecting the target core. See the following figures as reference.

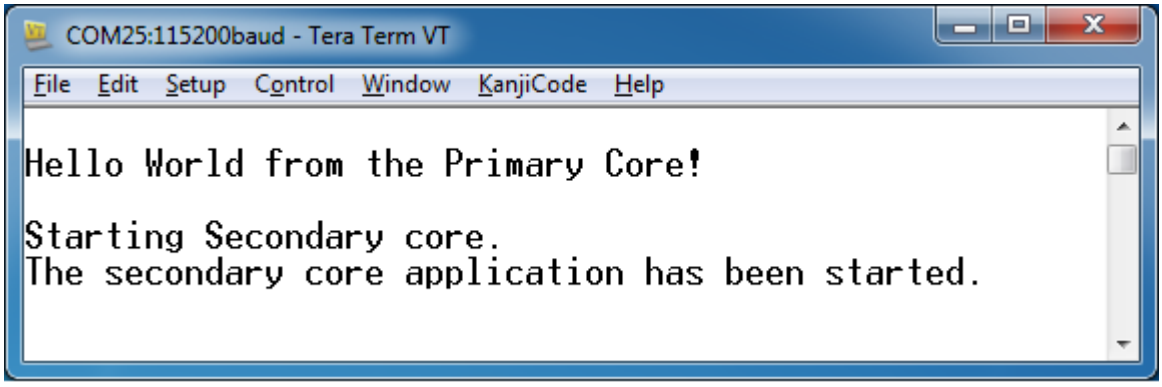



Probes discovered
Connect to target: LPC54114J256
The following probes have been disabled in the preferences:
P&E Micro probes SEGGER J-Link probes
Available attached probes
Name
Serial number/ID
Type
Manufa...
IDE Debug Mode
LPC-LINK2 CMSIS-DAP V5.134
A000000002
LinkServe
NXP Semi
Non-Stop
Supported Probes (tick/untick to enable/disable)
MCUXpresso IDE LinkServer (inc. CMSIS-DAP) probes
P&E Micro probes
SEGGER J-Link probes
Probe search options
Search again
Remember my selection (for this Launch configuration)
OK
Cancel


SWD Configuration
2 available SWD Devices detected.
Target 'Cortex-M4' has been selected, but it may be incompatible!
Device
Name
TAP Id
Details
0
Cortex-M4
0x2ba01477
APID:24770011
1
Cortex-M0+
0x2ba01477
APID:24770011
OK
Cancel

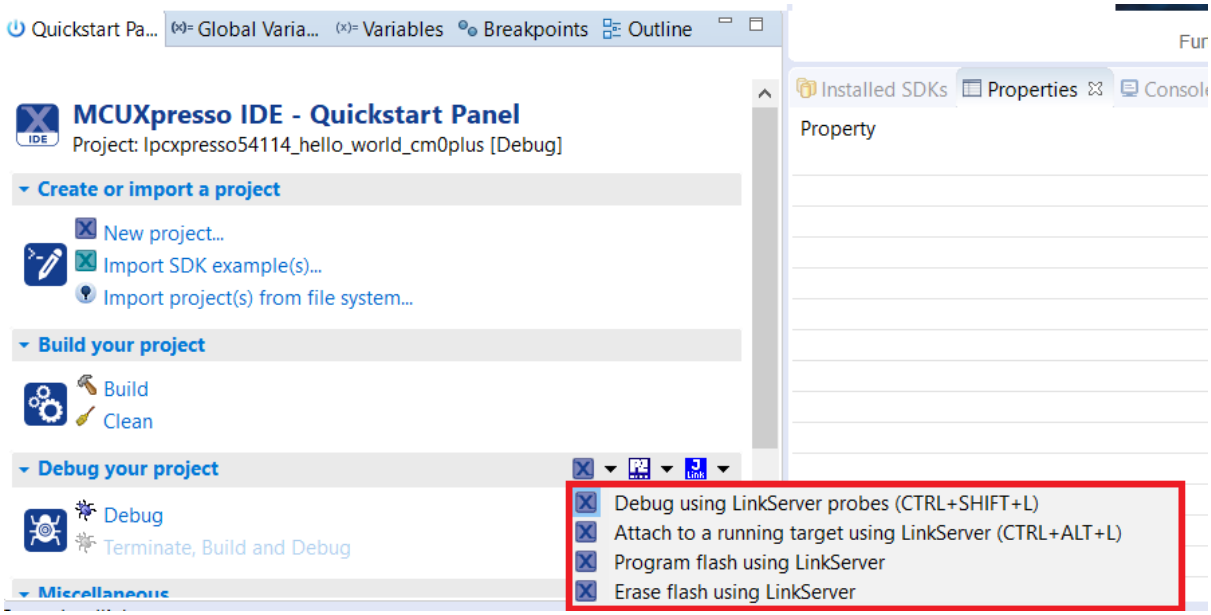


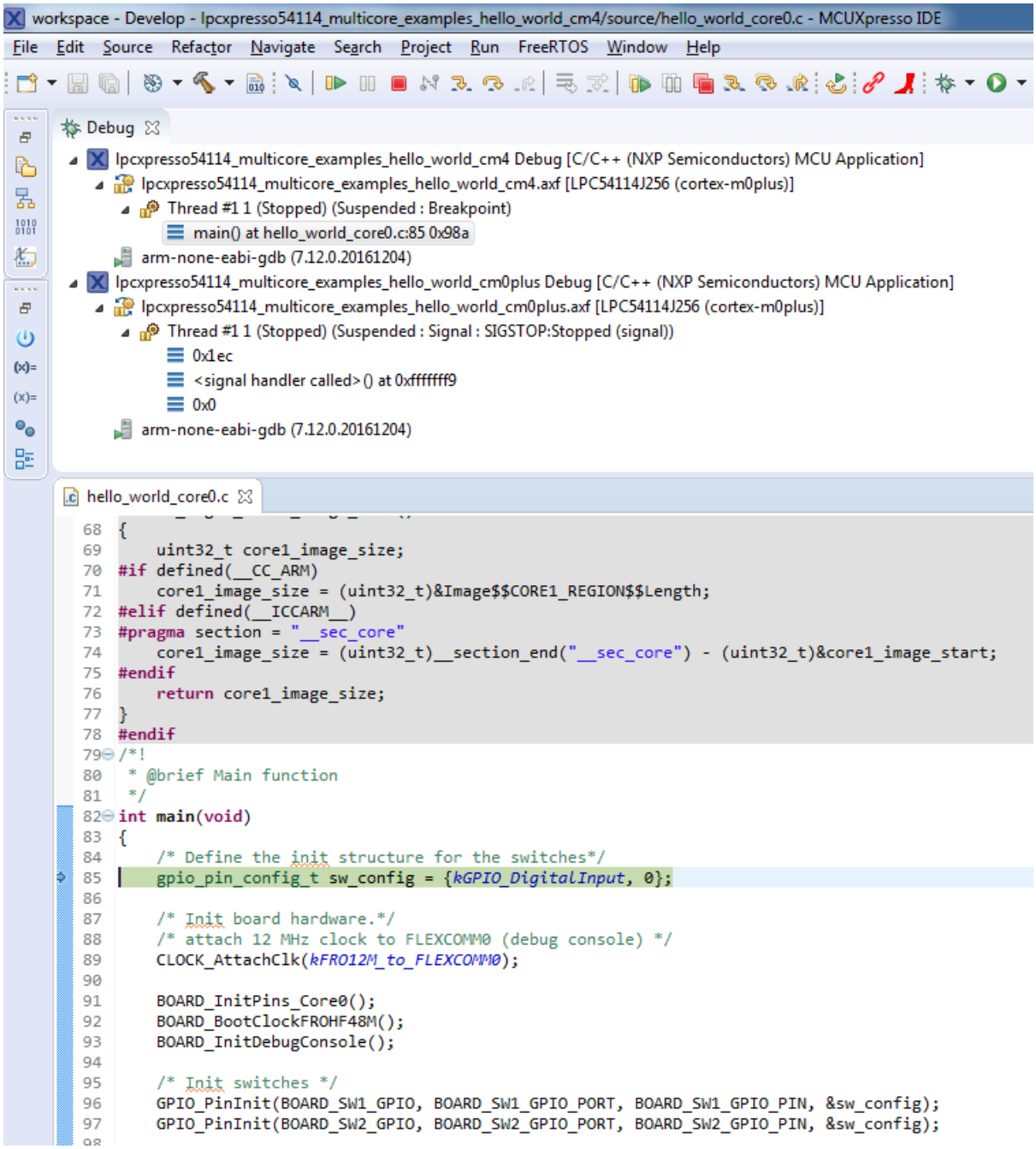
After clicking the “Resume All Debug sessions” button, the hello_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



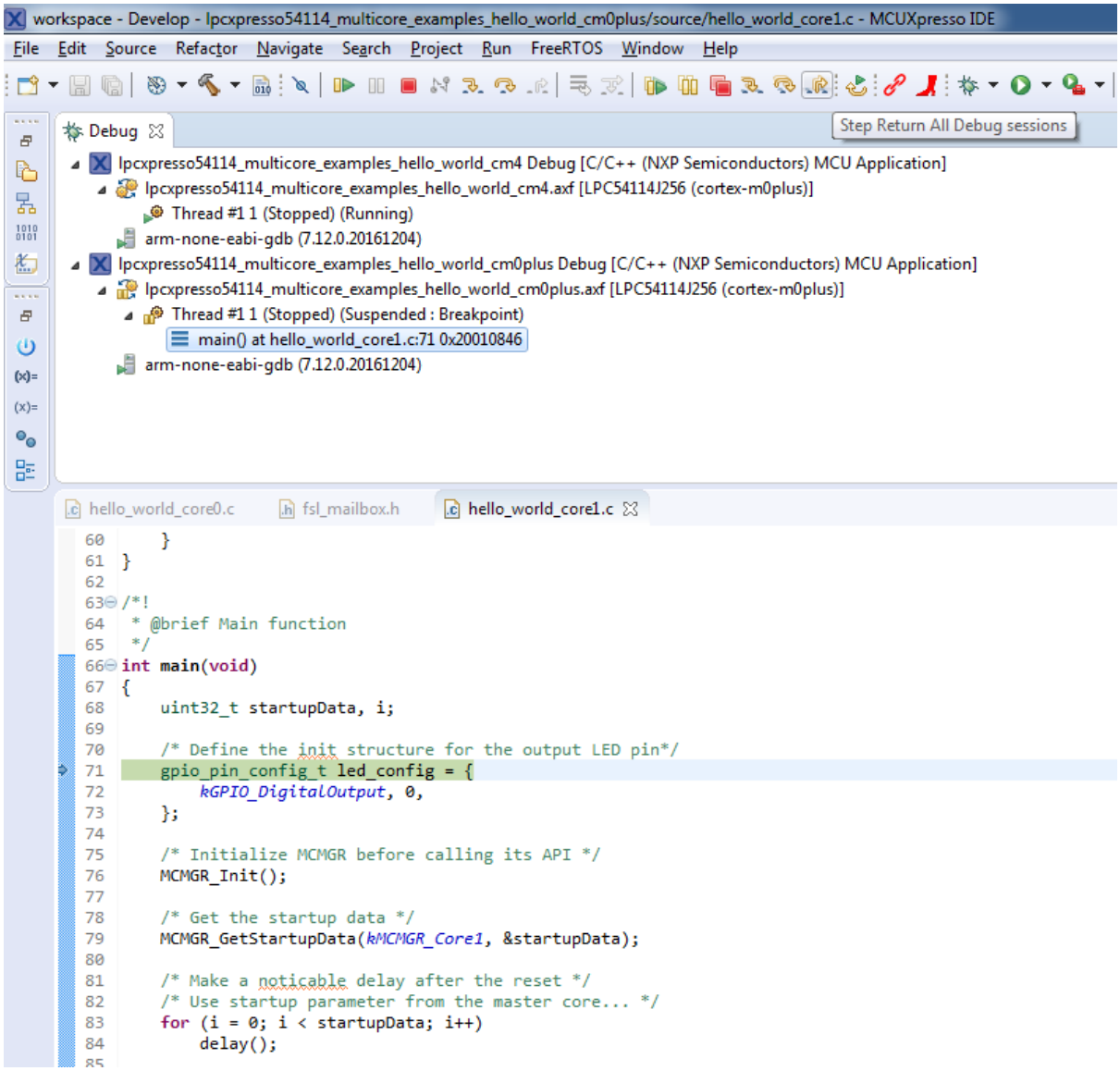
An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and running correctly. It is also possible to debug both sides of the multicore application in parallel. After creating the debug session for the primary core, perform same steps also for the auxiliary core application. Highlight the lpcxpresso54114_multicore_examples_hello_world_cm0plus project (multicore slave project) in the Project Explorer. On the Quickstart Panel, click “Debug ‘lpcxpresso54114_multicore_examples_hello_world_cm0plus’ [Debug]” to launch the second debug

session.

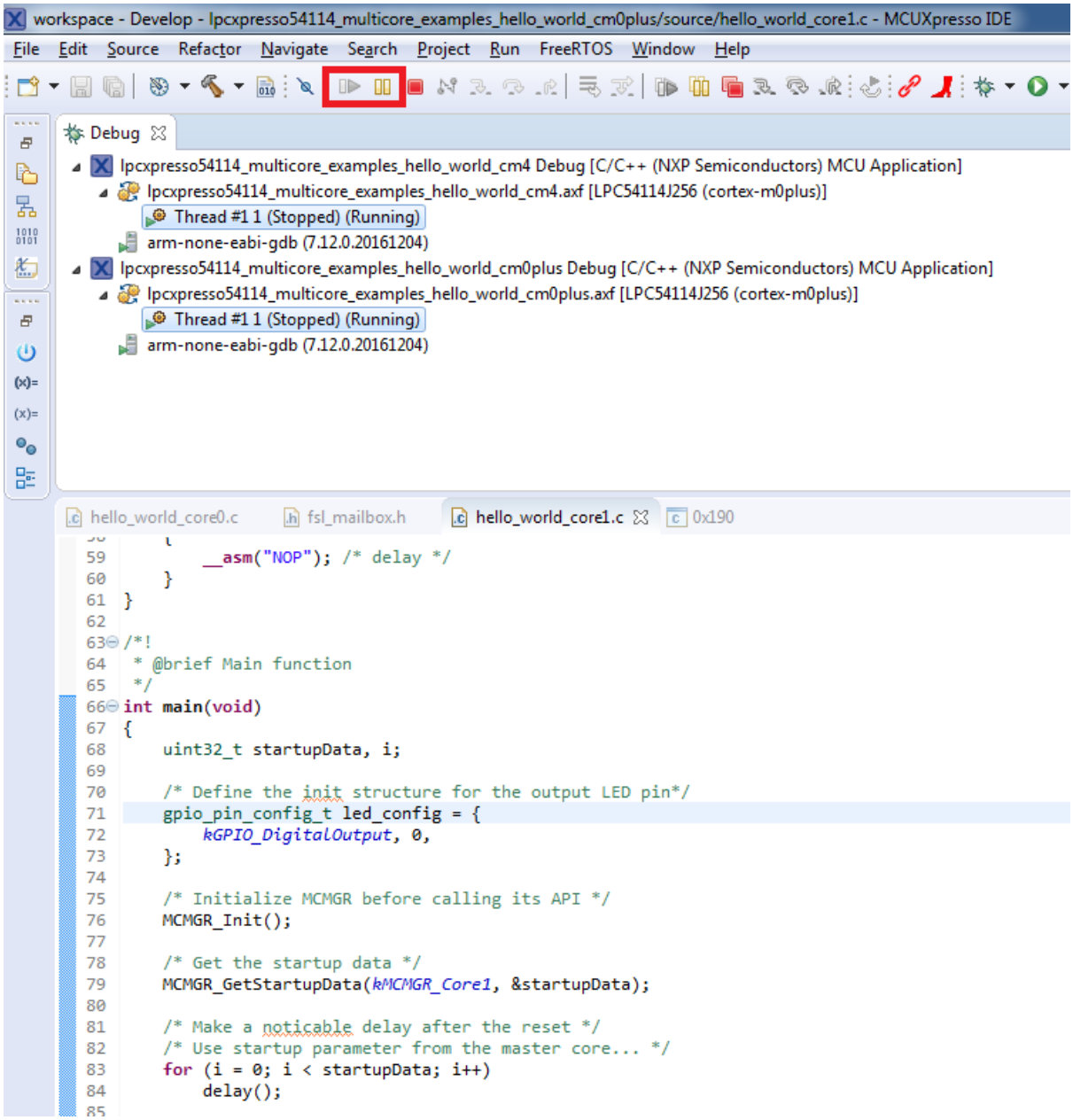

Quickstart Pa...
Global Varia...
Variables
Breakpoints
Outline
MCUXpresso IDE - Quickstart Panel
Project: lpcxpresso54114_hello_world_cm0plus [Debug]
Create or import a project
New project...
Import SDK example(s)...
Import project(s) from file system...
Build your project
Build
Clean
Debug your project
Debug
Terminate, Build and Debug
Miscellaneous
Installed SDKs
Properties
Property
Debug using LinkServer probes (CTRL+SHIFT+L)
Attach to a running target using LinkServer (CTRL+ALT+L)
Program flash using LinkServer
Erase flash using LinkServer

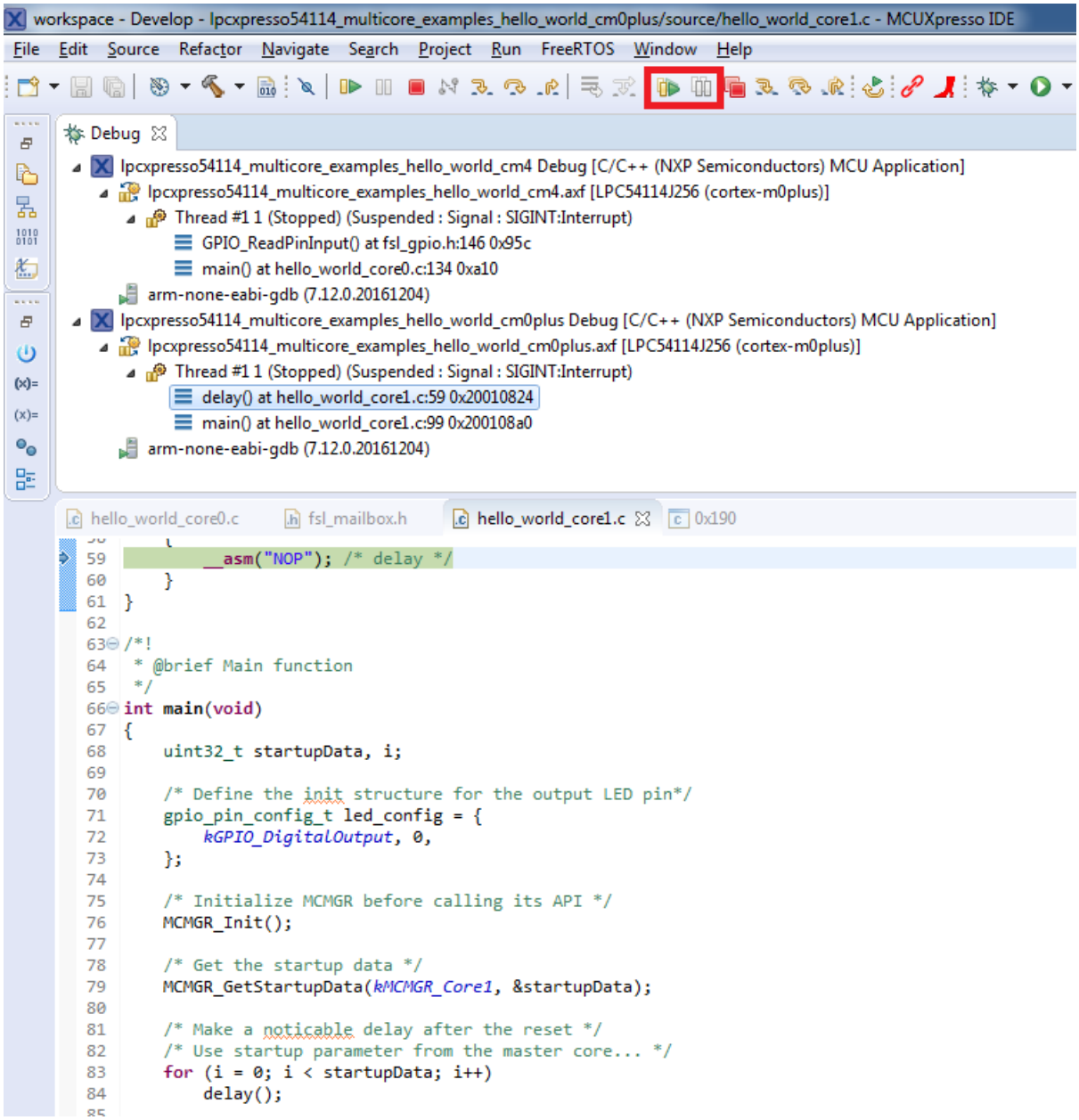


Now, the two debug sessions should be opened, and the debug controls can be used for both debug sessions depending on the debug session selection. Keep the primary core debug session selected by clicking the “Resume” button. The hello_world multicore application then starts running. The primary core application starts the auxiliary core application during runtime, and the auxiliary core application stops at the beginning of the main() function. The debug session of the auxiliary core application is highlighted. After clicking the “Resume” button, it is applied to the auxiliary core debug session. Therefore, the auxiliary core application continues its execution.



At this point, it is possible to suspend and resume individual cores independently. It is also possible to make synchronous suspension and resumption of both the cores. This is done either by selecting both opened debug sessions (multiple selections) and clicking the “Suspend” / “Resume” control button, or just using the “Suspend All Debug sessions” and the “Resume All Debug sessions” buttons.

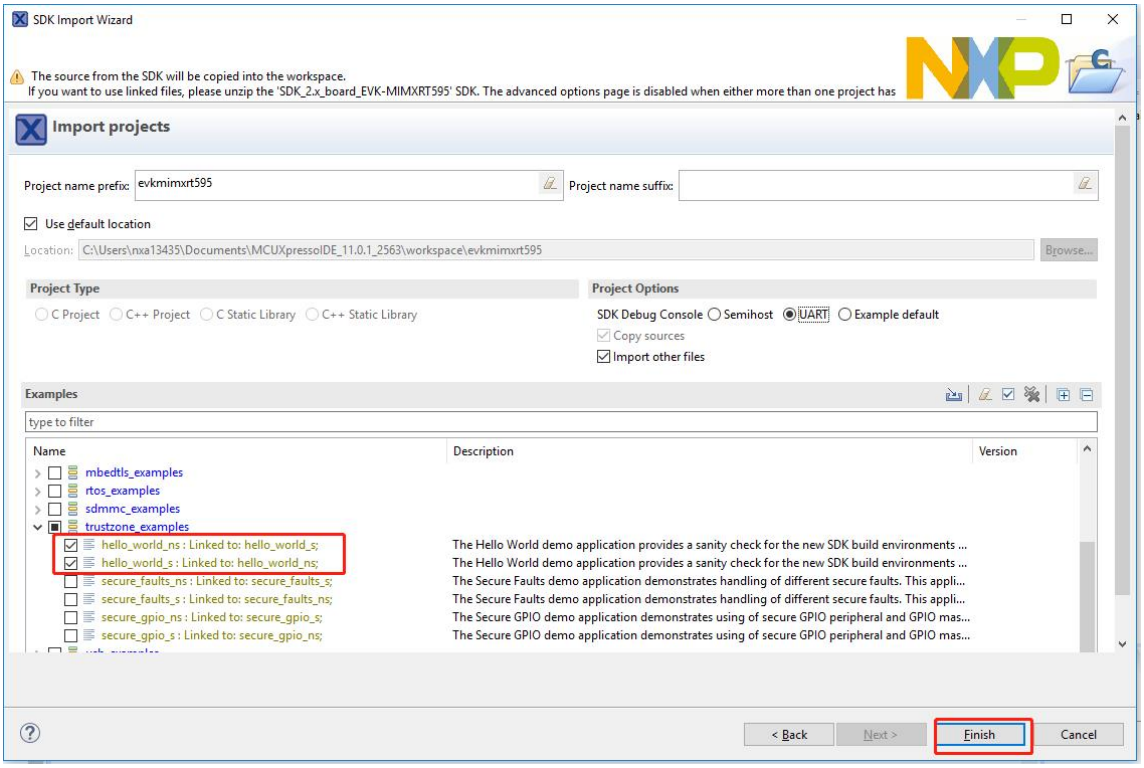

workspace - Develop - lpcxpresso54114_multicore_examples_hello_world_cm0plus/source/hello_world_core1.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project Run FreeRTOS Window Help
Debug
lpcxpresso54114_multicore_examples_hello_world_cm4 Debug [C/C++ (NXP Semiconductors) MCU Application]
lpcxpresso54114_multicore_examples_hello_world_cm4.axf [LPC54114J256 (cortex-m0plus)]
Thread #1 1 (Stopped) (Running)
arm-none-eabi-gdb (7.12.0.20161204)
lpcxpresso54114_multicore_examples_hello_world_cm0plus Debug [C/C++ (NXP Semiconductors) MCU Application]
lpcxpresso54114_multicore_examples_hello_world_cm0plus.axf [LPC54114J256 (cortex-m0plus)]
Thread #1 1 (Stopped) (Running)
arm-none-eabi-gdb (7.12.0.20161204)
hello_world_core0.c
fsl_mailbox.h
hello_world_core1.c
0x190
59 __asm("NOP"); /* delay */
60 }
61 }
62
63 /*!
64 * @brief Main function
65 */
66 int main(void)
67 {
68 uint32_t startupData, i;
69
70 /* Define the init structure for the output LED pin*/
71 gpio_pin_config_t led_config = {
72 kGPIO_DigitalOutput, 0,
73 };
74
75 /* Initialize MCMGR before calling its API */
76 MCMGR_Init();
77
78 /* Get the startup data */
79 MCMGR_GetStartupData(kMCMGR_Core1, &startupData);
80
81 /* Make a noticable delay after the reset */
82 /* Use startup parameter from the master core... */
83 for (i = 0; i < startupData; i++)
84 delay();
85



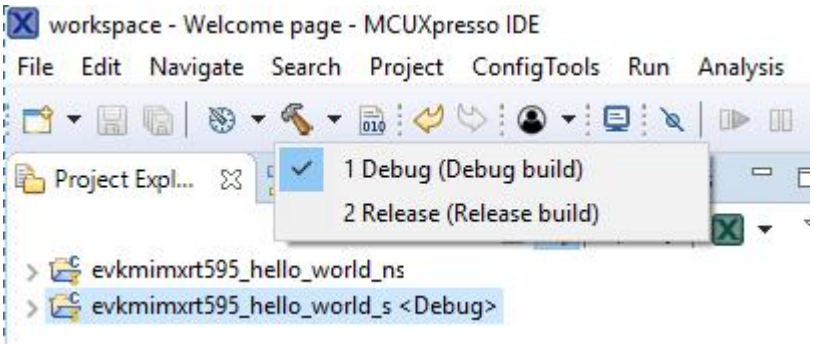
Build a TrustZone example application

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 MIMXRT595-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 MIMXRT595-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 **MIMXRT500** folder and select **MIMXRT595S**. Then, select **evkmimxrt595** 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**.

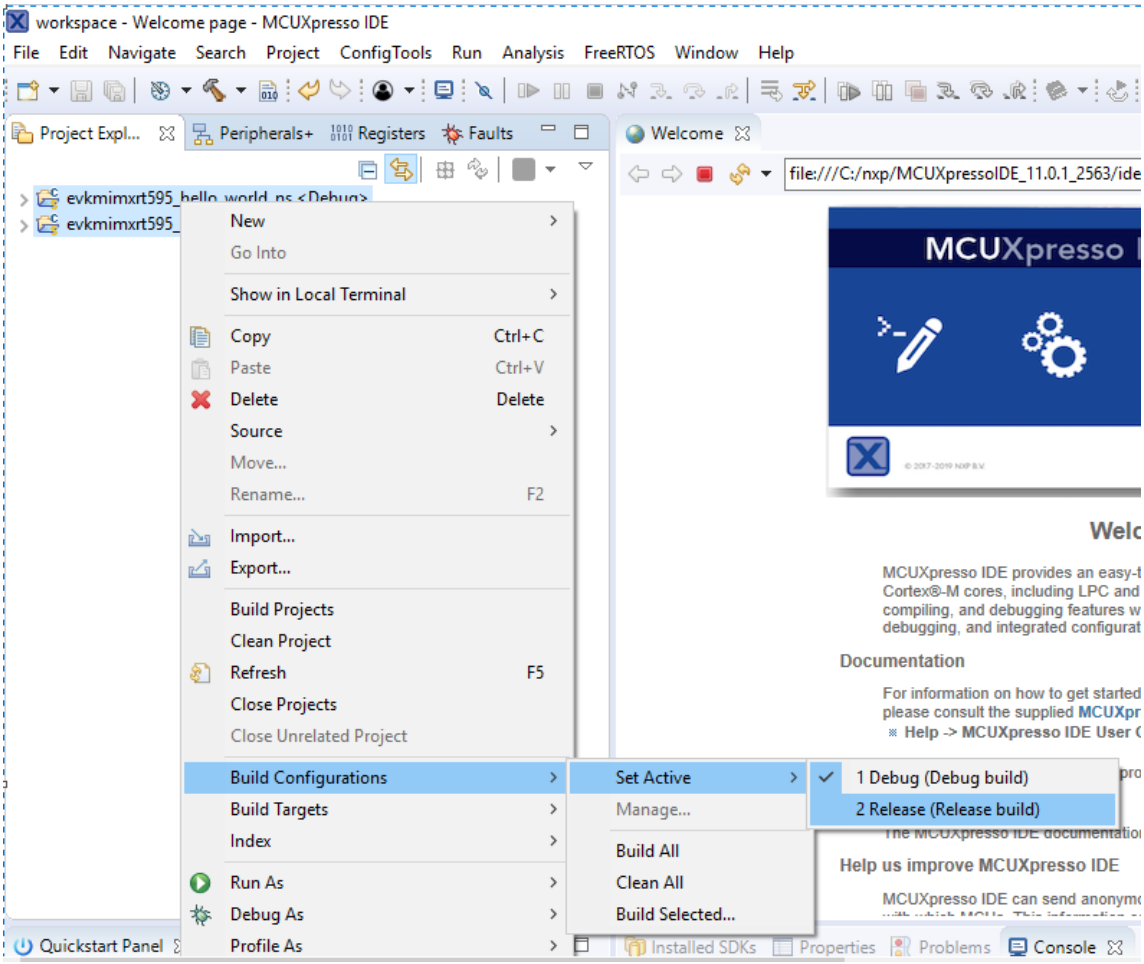


3. Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the evkmimxrt595_hello_world_s project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, **Debug** or **Release**, by clicking the downward facing arrow next to the hammer icon, as shown in following figure. 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.

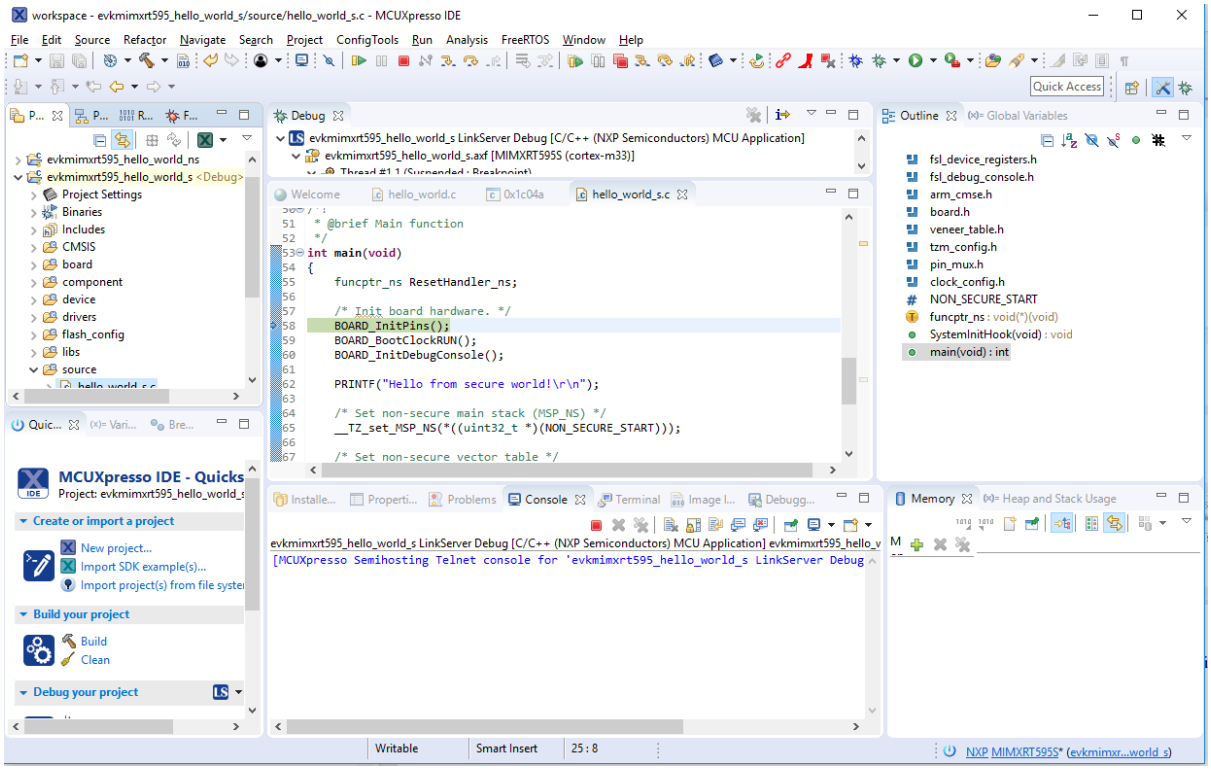
Note: When the **Release** build is requested, it is necessary to change the build configuration of both the secure and non-secure application projects first. To do this, select both projects in the Project Explorer view by clicking to select the first project, then using shift-click or control-click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



Run a TrustZone example application

To download and run the application, perform all steps as described in **Run an example application**. These steps are common for single core, and TrustZone applications, ensuring <board_name>_hello_world_s is selected for debugging.

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



Now, the TrustZone sessions 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 8.32.3 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*.

Build an example application

Do the following steps to build the hello_world example application.

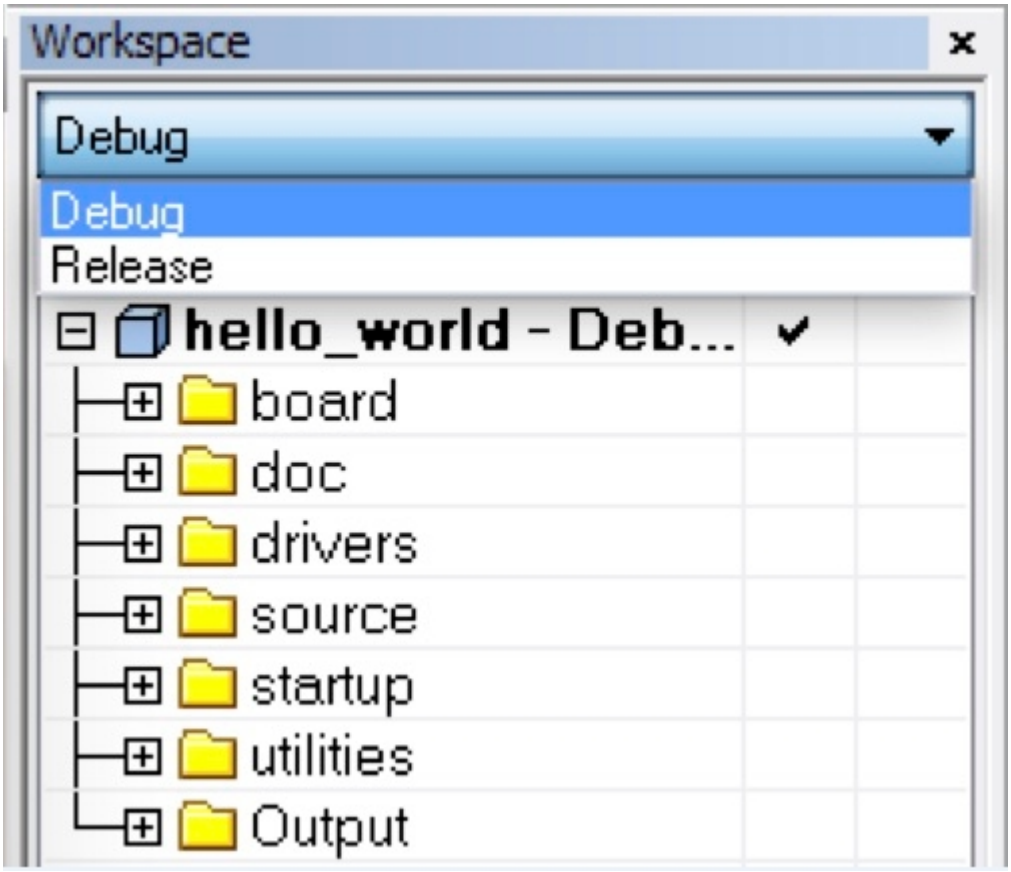
1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

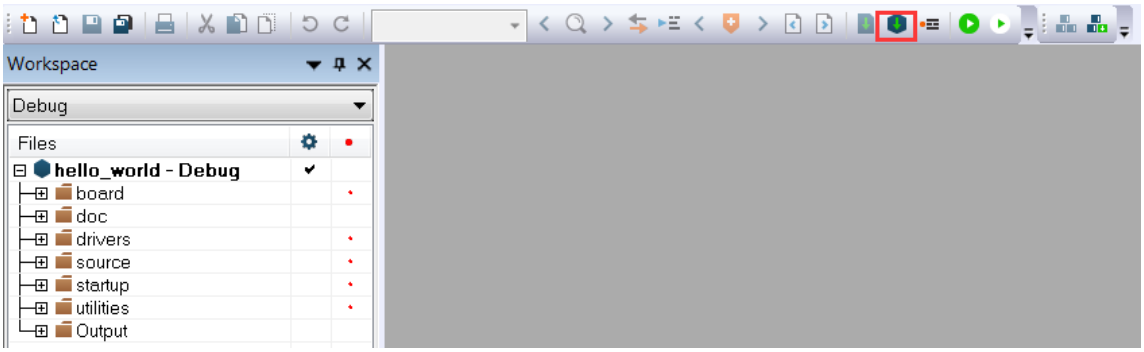
   Other example applications may have additional folders in their path.

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 following figure.



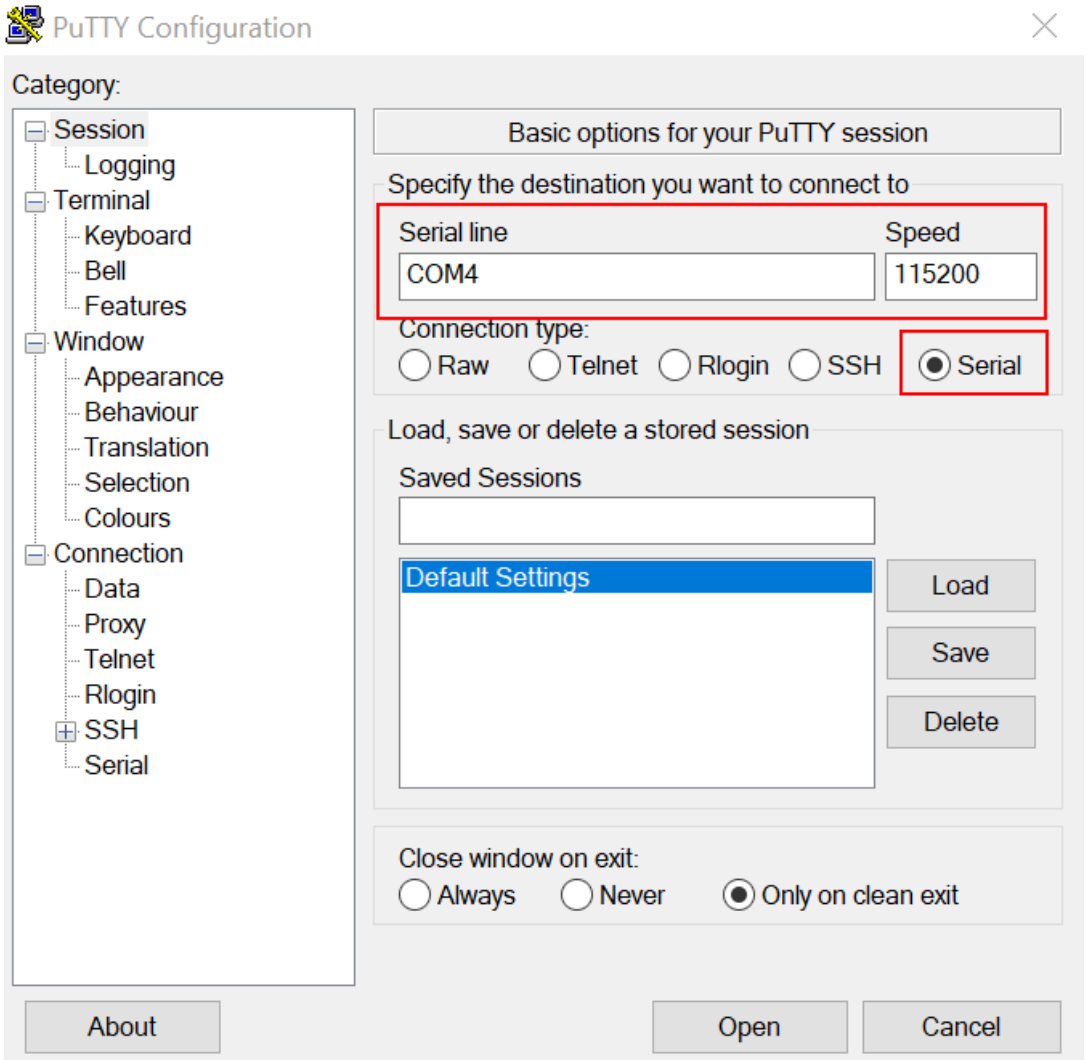
4. The build completes without errors.

Run an example application

To download and run the application, perform these steps:

1. Ensure the host driver for the debugger firmware has been installed. See *On-board debugger*.
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

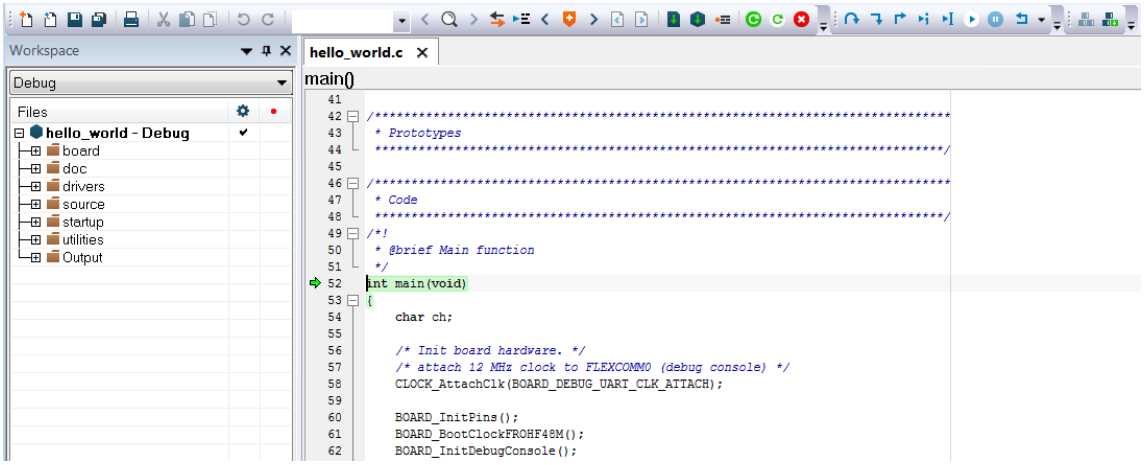
3. 8 data bits



4. 1 stop bit

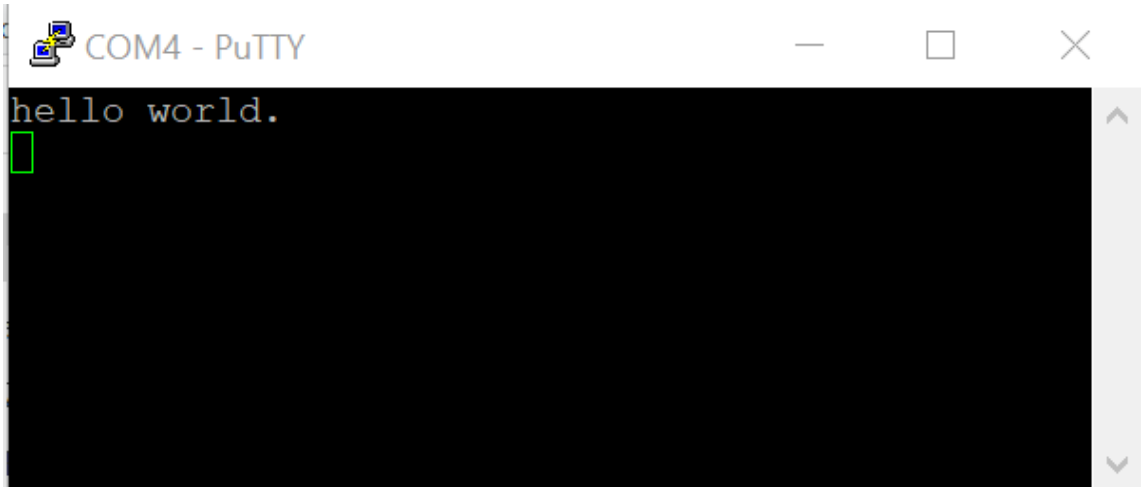
4. In IAR, click the **Download and Debug** button to download the application to the target.

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.



Build a multicore example application

This section describes the steps to build and run a dual-core application. The demo applications workspace files are located in this folder:

```
<install_dir>/boards/<board_name>/multicore_examples/<application_name>/<core_type>/iar
```

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

```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/iar/hello_world_cm0plus.
↪eww
```

```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm4/iar/hello_world_cm4.eww
```

Build both applications separately by clicking the **Make** button. Build the application for the auxiliary core (cm0plus) first, because the primary core application project (cm4) must know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

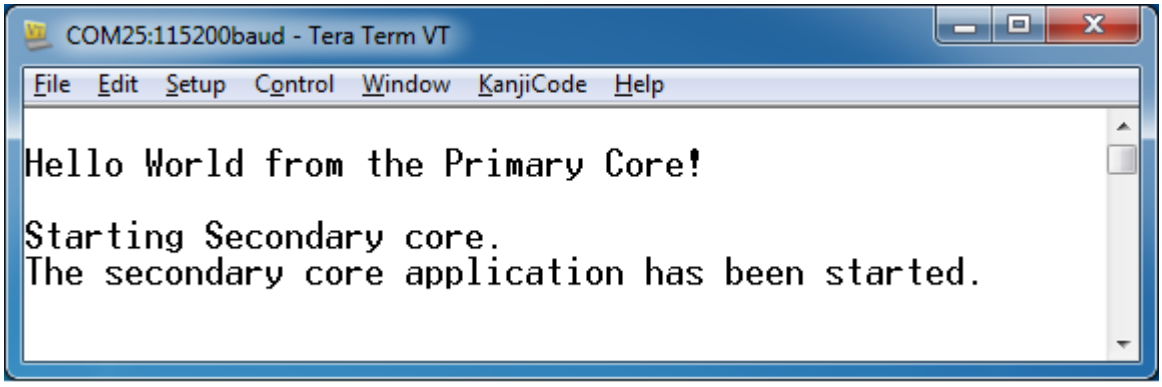
Run a multicore example application

The primary core debugger handles flashing both primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 4 as described in **Run an example application**. These steps are common for both single core and dual-core applications in IAR.

After clicking the “Download and Debug” button, the auxiliary core project is opened in the separate EWARM instance. Both the primary and auxiliary images are loaded into the device flash memory and the primary core application is executed. It stops at the default C language entry point in the *main()*function.

Run both cores by clicking the “Start all cores” button to start the multicore application.

During the primary core code execution, the auxiliary core is released from the reset. The hello_world multicore application is now running and a banner is displayed on the terminal. If this does not appear, check the terminal settings and connections.



An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and is running correctly. When both cores are running, use the "Stop all cores", and "Start all cores" control buttons to stop or run both cores simultaneously.

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/<board_name>/trustzone_examples/hello_world/hello_world_ns/iar/hello_world_
↪ns.eww
```

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

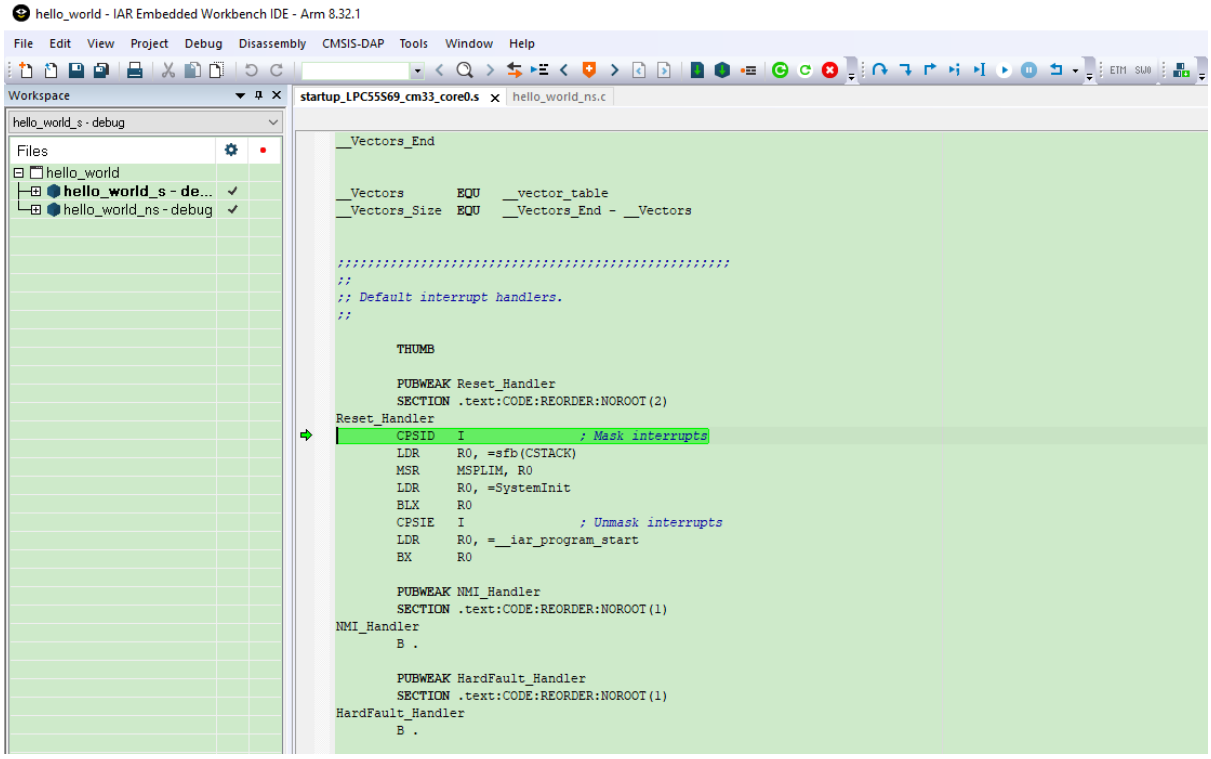
```
<install_dir>/boards/<board_name>/trustzone_examples/hello_world/hello_world_s/iar/hello_world.eww
```

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

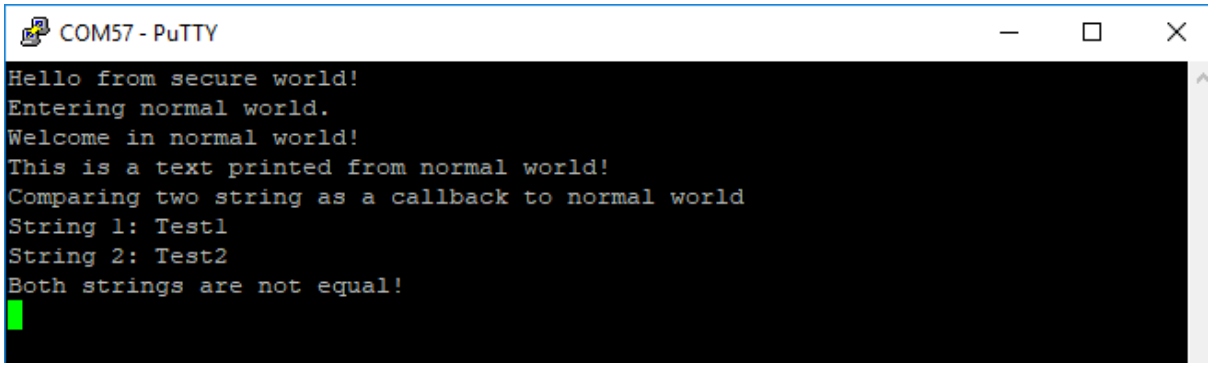
The secure project is configured to download both secure and non-secure output files, so debugging can be fully managed from the secure project. To download and run the TrustZone application, switch to the secure application project and perform steps 1 – 4 as described in **Run an example application**. 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 Reset_Handler function.

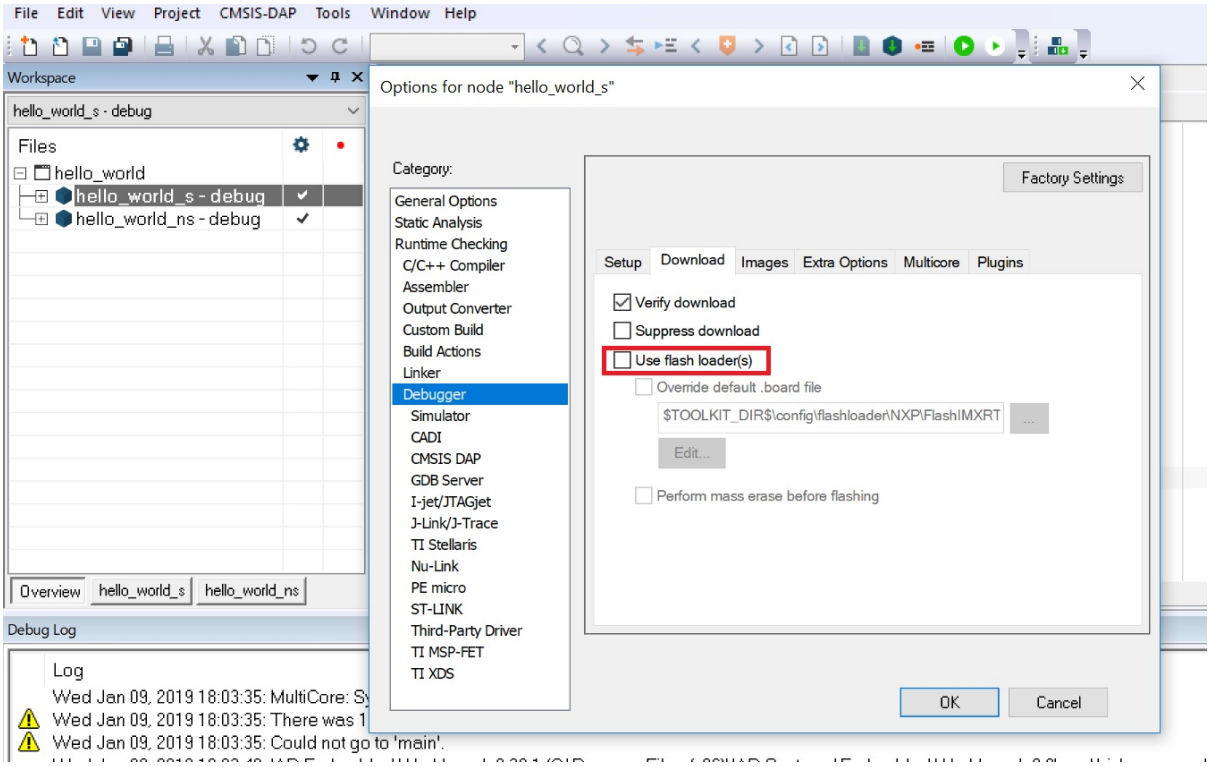


Run the code by clicking **Go** to start the application.

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.MXRT500.

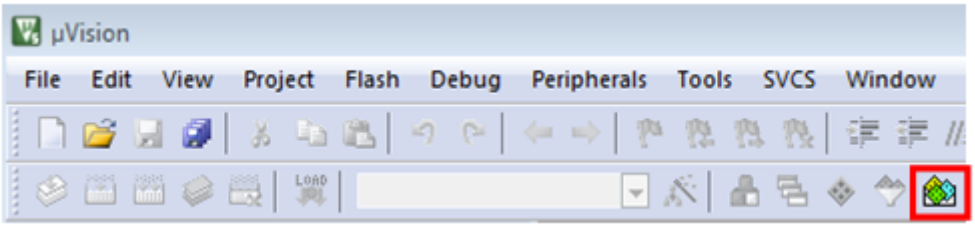


Run a demo using Keil MDK/µVision This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK.

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 appropriate CMSIS pack.

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



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

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:

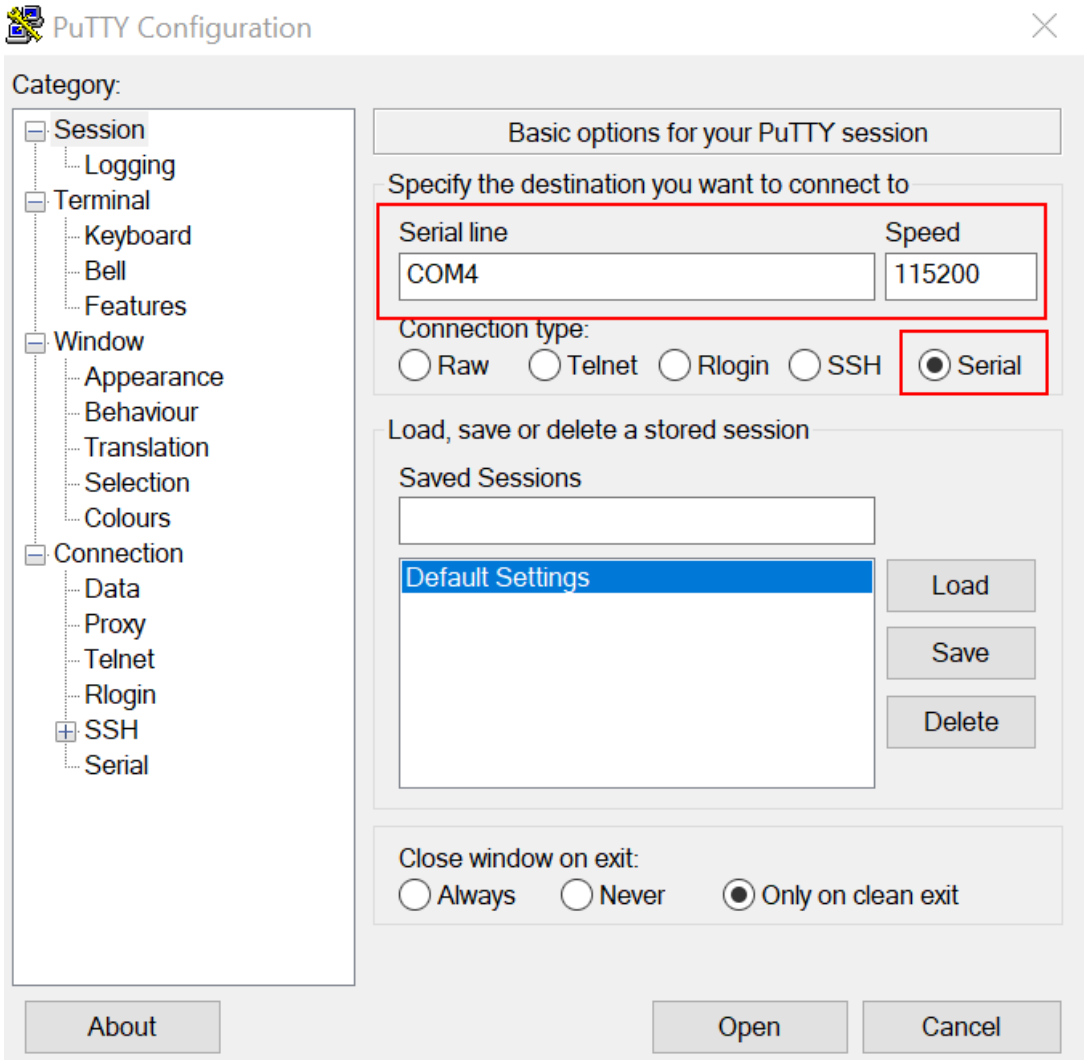
2. To build the demo project, select **Rebuild**, highlighted in red.

3. The build completes without errors.

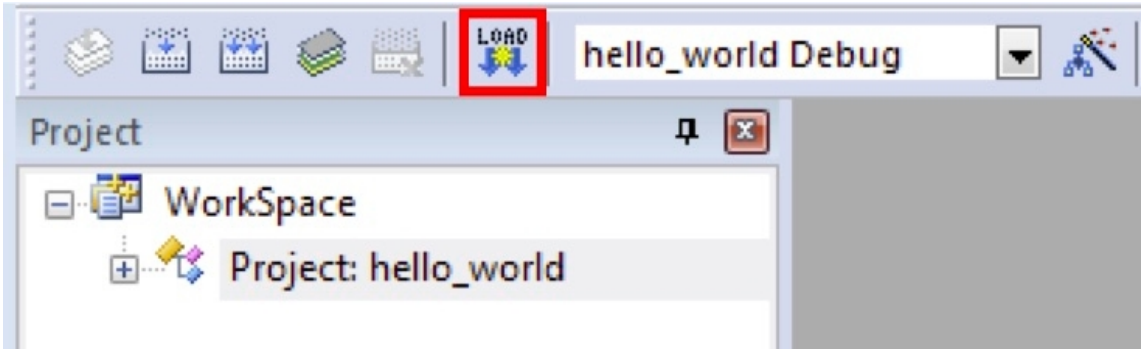
Run an example application

To download and run the application, perform these steps:

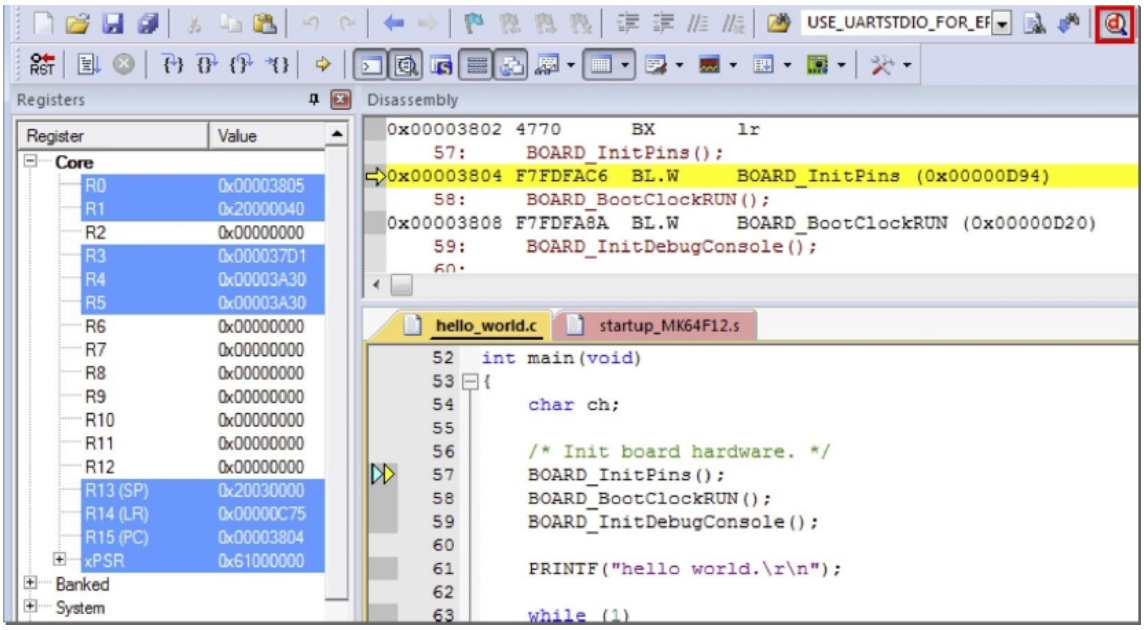
1. Ensure the host driver for the debugger firmware has been installed. See *On-board debugger*.
2. Connect the development platform to your PC via USB cable using USB connector.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm and connect to the debug serial port number (to determine the COM port number, see *How to determine COM port*. Configure the terminal with these settings:
    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



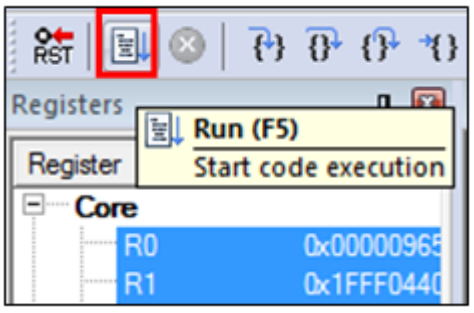
4. In µVision, after the application is built, click the **Download** button to download the application to the target.



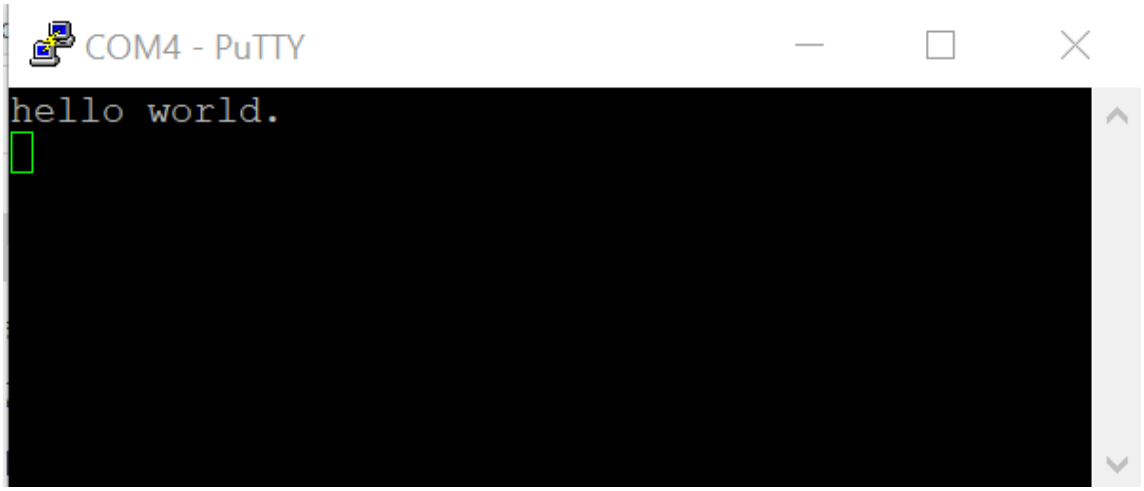
5. After clicking the **Download** button, the application downloads to the target and is running. To debug the application, click the **Start/Stop Debug Session** button, highlighted in red.



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



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



Build a multicore example application

This section describes the steps to build and run a dual-core application. The demo applications workspace files are located in this folder:

```
<install_dir>/boards/<board_name>/multicore_examples/<application_name>/<core_type>/mdk
```

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

```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm0plus/mdk/hello_world_
↪cm0plus.uvmpw
```

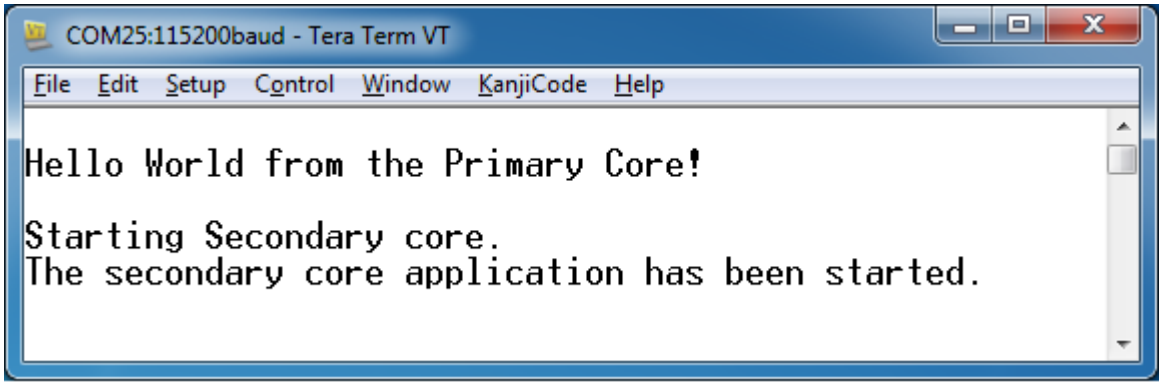
```
<install_dir>/boards/lpcxpresso54114/multicore_examples/hello_world/cm4/mdk/hello_world_cm4.uvmpw
```

Build both applications separately by clicking the **Rebuild** button. Build the application for the auxiliary core (cm0plus) first because the primary core application project (cm4) must know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

Run a multicore example application

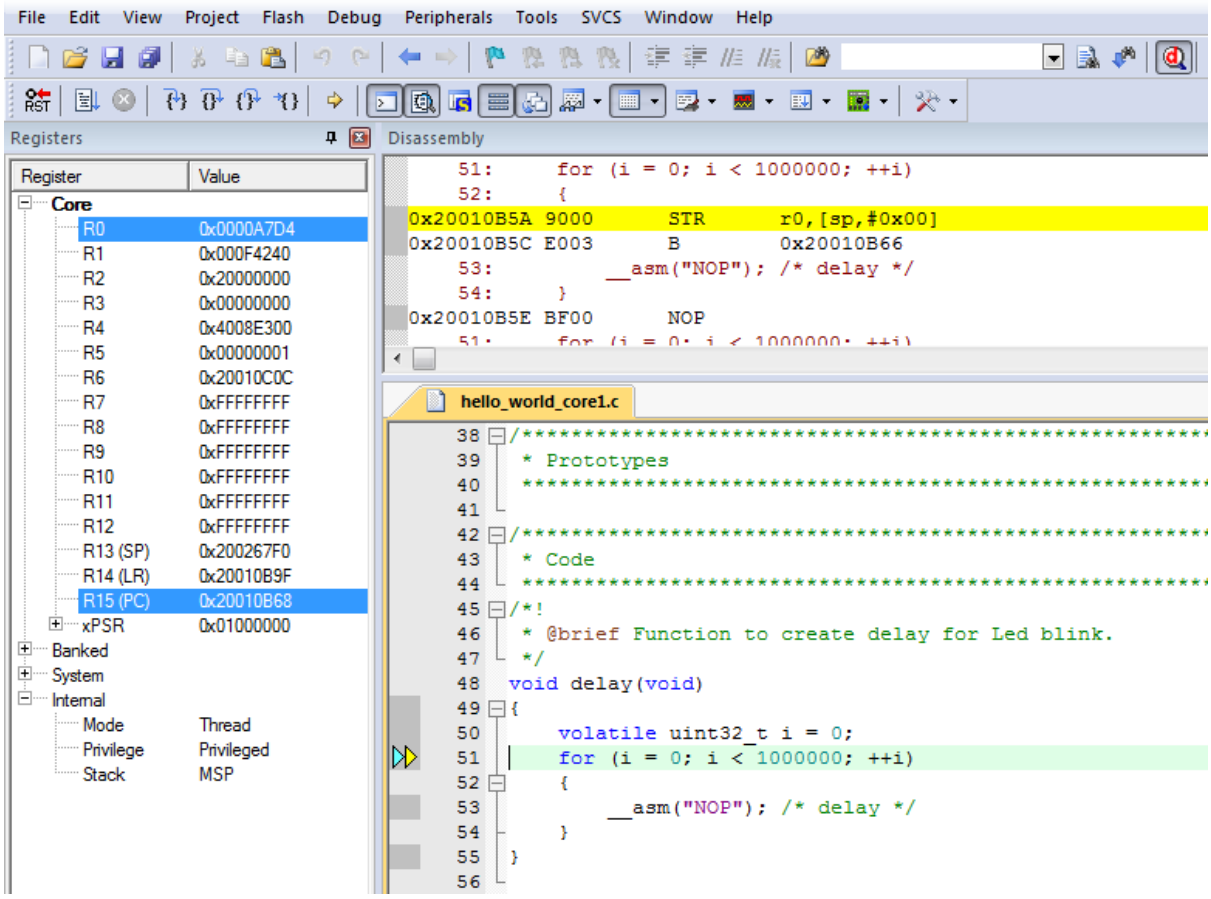
The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in **Run an example application**. These steps are common for both single-core and dual-core applications in µVision.

Both the primary and the auxiliary image is loaded into the device flash memory. After clicking the “Run” button, the primary core application is executed. During the primary core code execution, the auxiliary core is released from the reset. The hello_world multicore application is now running and a banner is displayed on the terminal. If this does not appear, check your terminal settings and connections.



An LED controlled by the auxiliary core starts flashing indicating that the auxiliary core has been released from the reset and is running correctly.

Attach the running application of the auxiliary core by opening the auxiliary core project in the second µVision instance and clicking the "Start/Stop Debug Session" button. After this, the second debug session is opened and the auxiliary core application can be debugged.



Arm describes multicore debugging using the NXP LPC54114 Cortex-M4/M0+ dual-core processor and Keil uVision IDE in Application Note 318 at www.keil.com/appnotes/docs/apnt_318.asp. The associated video can be found here.

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/<board_name>/trustzone_examples/hello_world/hello_world_ns/mdk/hello_world_
↪ns.uvmpw
```

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

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

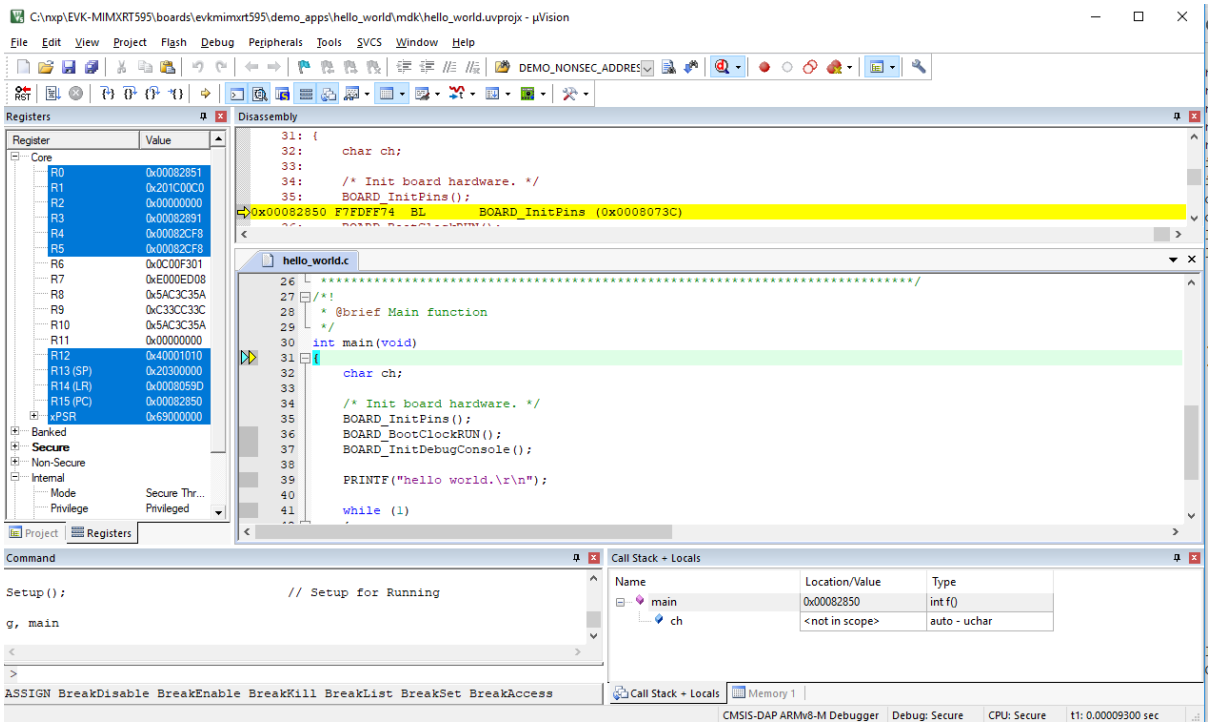
This project hello_world.uvmpw contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another.

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

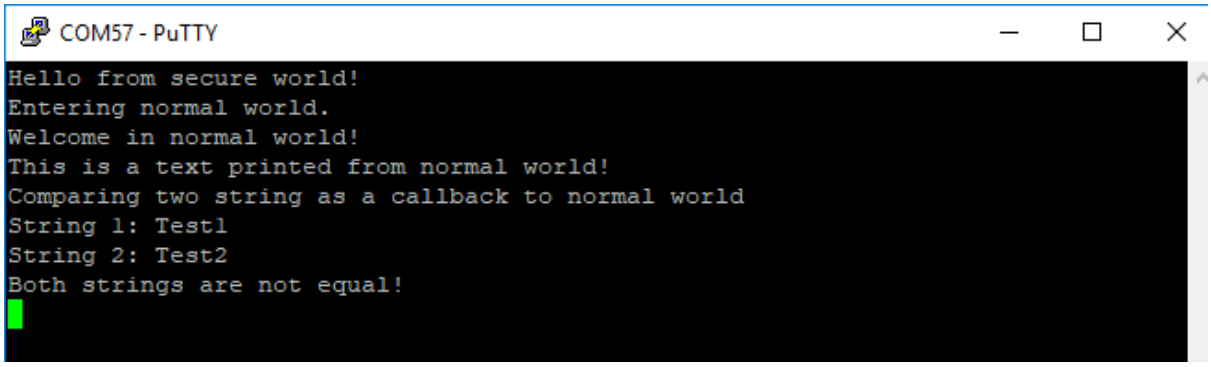
The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

To download and run the TrustZone application, switch to the secure application project and perform steps as described in **Run an 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 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.

MCUXpresso Config Tools MCUXpresso Config Tools can help configure the processor and generate initialization code for the on chip peripherals. The tools are able to modify any existing example project, or create a new configuration for the selected board or processor. The generated code is designed to be used with MCUXpresso SDK version 24.12.00 or later.

Following table describes the tools included in the MCUXpresso Config Tools.

Config Tool	Description	Image
Pins tool	For configuration of pin routing and pin electrical properties.	
Clock tool	For system clock configuration	
Peripherals tools	For configuration of other peripherals	
TEE tool	Configures access policies for memory area and peripherals helping to protect and isolate sensitive parts of the application.	
Device Configuration tool	Configures Device Configuration Data (DCD) contained in the program image that the Boot ROM code interprets to set up various on-chip peripherals prior to the program launch.	

MCUXpresso Config Tools can be accessed in the following products:

- **Integrated** in the MCUXpresso IDE. Config tools are integrated with both compiler and debugger which makes it the easiest way to begin the development.
- **Standalone version** available for download from www.nxp.com/mcuxpresso. Recommended for customers using IAR Embedded Workbench, Keil MDK µVision, or Arm GCC.

- **Online version** available on mcuxpresso.nxp.com. Recommended doing a quick evaluation of the processor or use the tool without installation.

Each version of the product contains a specific *Quick Start Guide* document MCUXpresso IDE Config Tools installation folder that can help start your work.

How to determine COM port This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform. All NXP boards ship with a factory programmed, onboard debug interface, whether it is based on MCU-Link or the legacy OpenSDA, LPC-Link2, P&E Micro OSJTAG interface. To determine what your specific board ships with, see *Default debug interfaces*.

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

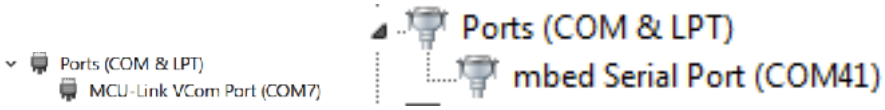
```
$ dmesg | grep "ttyUSB"
  [503175.307873] usb 3-12: cp210x converter now attached to ttyUSB0
  [503175.309372] usb 3-12: cp210x converter now attached to ttyUSB1
```

   There are two ports, one is for core0 debug console and the other is for core1.

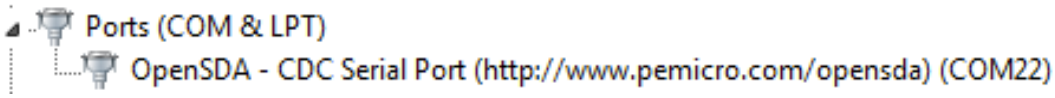
2. **Windows**: To determine the COM port open Device Manager in the Windows operating system. Click the **Start** menu and type **Device Manager** in the search bar.

   In the Device Manager, expand the **Ports (COM & LPT)** section to view the available ports. The COM port names are different for all the NXP boards.

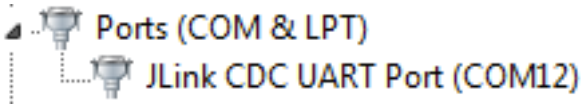
   1. **CMSIS-DAP/mbed/DAPLink** interface:

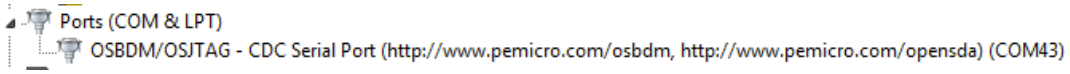
   2. **P&E Micro**:

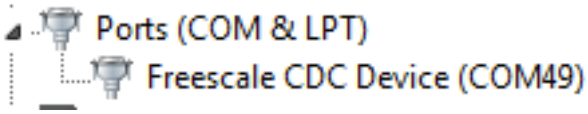
   3. **J-Link**:

   4. **P&E Micro OSJTAG**:

   5. **MRB-KW01**:

On-board Debugger This section describes the on-board debuggers used on NXP development boards.

On-board debugger MCU-Link

MCU-Link is a powerful and cost effective debug probe that can be used seamlessly with MCUXpresso IDE, and is also compatible with 3rd party IDEs that support CMSIS-DAP protocol. MCU-Link also includes a USB to UART bridge feature (VCOM) that can be used to provide a serial connection between the target MCU and a host computer. MCU-Link features a high-speed USB interface for high performance debug. MCU-Link is compatible with Windows, MacOS and Linux. A free utility from NXP provides an easy way to install firmware updates.

On-board MCU-Link debugger supports CMSIS-DAP and J-Link firmware. See the table in *Default debug interfaces* to determine the default debug interface that comes loaded on your specific hardware platform.

The corresponding host driver must be installed before debugging.

- For boards with CMSIS-DAP firmware, 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.
- If using J-Link with either a standalone debug pod or MCU-Link, install the J-Link software (drivers and utilities) from www.segger.com/jlink-software.html.

Updating MCU-Link firmware

This firmware in this debug interface may be updated using the host computer utility called MCU-Link. 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), MCU-Link 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 MCU-Link utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto MCU-Link or NXP boards. The utility can be downloaded from MCU-Link.

These steps show how to update the debugger firmware on your board for Windows operating system.

1. Install the MCU-Link 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 MCU-Link installation directory (<MCU-Link install dir>).
    1. To program CMSIS-DAP debug firmware: <MCU-Link install dir>/scripts/program_CMSIS
    2. To program J-Link debug firmware: <MCU-Link 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.

On-board debugger LPC-Link

LPC-Link 2 is an extensible debug probe that can be used seamlessly with MCUXpresso IDE, and is also compatible with 3rd party IDEs that support CMSIS-DAP protocol. MCU-Link also includes a USB to UART bridge feature (VCOM) that can be used to provide a serial connection between

the target MCU and a host computer. LPC-Link 2 is compatible with Windows, MacOS and Linux. A free utility from NXP provides an easy way to install firmware updates.

On-board LPC-Link 2 debugger supports CMSIS-DAP and J-Link firmware. See the table in *Default debug interfaces* to determine the default debug interface that comes loaded on your specific hardware platform.

The corresponding host driver must be installed before debugging.

- For boards with CMSIS-DAP firmware, 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.
- If using J-Link with either a standalone debug pod or MCU-Link, install the J-Link software (drivers and utilities) from www.segger.com/jlink-software.html.

Updating LPC-Link 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 LPCScrypt.

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 (LPCScrypt, 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.

On-board debugger OpenSDA

OpenSDA/OpenSDAv2 is a serial and debug adapter that is built into several NXP evaluation boards. It provides a bridge between your computer (or other USB host) and the embedded target processor, which can be used for debugging, flash programming, and serial communication, all over a simple USB cable.

The difference is the firmware implementation: OpenSDA: Programmed with the proprietary P&E Micro developed bootloader. P&E Micro is the default debug interface app. OpenSDAv2: Programmed with the open-sourced CMSIS-DAP/mbed bootloader. CMSIS-DAP is the default debug interface app.

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

The corresponding host driver must be installed before debugging.

- For boards with CMSIS-DAP firmware, 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.

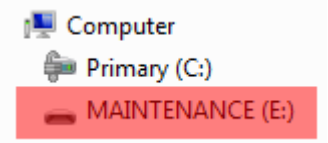
Updating OpenSDA firmware

Any NXP hardware platform that comes with an OpenSDA-compatible debug interface has the ability to update the OpenSDA firmware. This typically means to switch from the default application (either CMSIS-DAP or P&E Micro) to a SEGGER J-Link. This section contains the steps to switch the OpenSDA firmware to a J-Link interface. However, the steps can be applied to restoring the original image also. 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.
- CMSIS-DAP: CMSIS-DAP OpenSDA firmware is available at www.nxp.com/opensda.
- 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.

On-board debugger Multilink

An on-board Multilink debug circuit provides a JTAG interface and a power supply input through a single micro-USB connector. It is a hardware interface that allows PC software to debug and program a target processor through its debug port.

The host driver must be installed before debugging.

- See PE micro to download and install the P&E Micro Hardware Interface Drivers package.

On-board debugger OSJTAG

An on-board OSJTAG debug circuit provides a JTAG interface and a power supply input through a single micro-USB connector. It is a hardware interface that allows PC software to debug and program a target processor through its debug port.

The host driver must be installed before debugging.

- See PE micro to download and install the P&E Micro Hardware Interface Drivers package.

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

Hardware platform	Default debugger firmware	On-board debugger probe
EVK-MCIMX7ULP	N/A	N/A
EVK-MIMX8MM	N/A	N/A
EVK-MIMX8MN	N/A	N/A
EVK-MIMX8MNDDR3L	N/A	N/A
EVK-MIMX8MP	N/A	N/A
EVK-MIMX8MQ	N/A	N/A
EVK-MIMX8ULP	N/A	N/A
EVK-MIMXRT1010	CMSIS-DAP	LPC-Link2
EVK-MIMXRT1015	CMSIS-DAP	LPC-Link2
EVK-MIMXRT1020	CMSIS-DAP	LPC-Link2
EVK-MIMXRT1064	CMSIS-DAP	LPC-Link2
EVK-MIMXRT595	CMSIS-DAP	LPC-Link2
EVK-MIMXRT685	CMSIS-DAP	LPC-Link2
EVK9-MIMX8ULP	N/A	N/A
EVKB-IMXRT1050	CMSIS-DAP	LPC-Link2
FRDM-K22F	CMSIS-DAP	OpenSDA v2
FRDM-K32L2A4S	CMSIS-DAP	OpenSDA v2
FRDM-K32L2B	CMSIS-DAP	OpenSDA v2

continues on next page

Table 1 – continued from previous page

Hardware platform	Default debugger firmware	On-board debugger probe
FRDM-K32L3A6	CMSIS-DAP	OpenSDA v2
FRDM-KE02Z40M	P&E Micro	OpenSDA v1
FRDM-KE15Z	CMSIS-DAP	OpenSDA v2
FRDM-KE16Z	CMSIS-DAP	OpenSDA v2
FRDM-KE17Z	CMSIS-DAP	OpenSDA v2
FRDM-KE17Z512	CMSIS-DAP	MCU-Link
FRDM-MCXA153	CMSIS-DAP	MCU-Link
FRDM-MCXA156	CMSIS-DAP	MCU-Link
FRDM-MCXA174	CMSIS-DAP	MCU-Link
FRDM-MCXA266	CMSIS-DAP	MCU-Link
FRDM-MCXA344	CMSIS-DAP	MCU-Link
FRDM-MCXA346	CMSIS-DAP	MCU-Link
FRDM-MCXA366	CMSIS-DAP	MCU-Link
FRDM-MCXA577	CMSIS-DAP	MCU-Link
FRDM-MCXC041	CMSIS-DAP	MCU-Link
FRDM-MCXC162	CMSIS-DAP	MCU-Link
FRDM-MCXC242	CMSIS-DAP	MCU-Link
FRDM-MCXC353	CMSIS-DAP	MCU-Link
FRDM-MCXC444	CMSIS-DAP	MCU-Link
FRDM-MCXE247	CMSIS-DAP	MCU-Link
FRDM-MCXE31B	CMSIS-DAP	MCU-Link
FRDM-MCXN236	CMSIS-DAP	MCU-Link
FRDM-MCXN947	CMSIS-DAP	MCU-Link
FRDM-MCXN947T	CMSIS-DAP	MCU-Link
FRDM-MCXW23	CMSIS-DAP	MCU-Link
FRDM-MCXW71	CMSIS-DAP	MCU-Link
FRDM-MCXW72	CMSIS-DAP	MCU-Link
FRDM-RW612	CMSIS-DAP	MCU-Link
FRDM-KW43	CMSIS-DAP	MCU-Link
FRDM-MCXW70	CMSIS-DAP	MCU-Link
KW43-LOC	CMSIS-DAP	MCU-Link
MCXW70-LOC	CMSIS-DAP	MCU-Link
IMX943-EVK	N/A	N/A
IMX95LP4XEVK-15	N/A	N/A
IMX95LPD5EVK-19	N/A	N/A
IMX95VERDINEVK	N/A	N/A
KW45B41Z-EVK	CMSIS-DAP	MCU-Link
KW45B41Z-LOC	CMSIS-DAP	MCU-Link
KW47-EVK	CMSIS-DAP	MCU-Link
KW47-LOC	CMSIS-DAP	MCU-Link
LPC845BREAKOUT	CMSIS-DAP	LPC-Link2
LPCXpresso51U68	CMSIS-DAP	LPC-Link2
LPCXpresso54628	CMSIS-DAP	LPC-Link2
LPCXpresso54S018	CMSIS-DAP	LPC-Link2
LPCXpresso54S018M	CMSIS-DAP	LPC-Link2
LPCXpresso55S06	CMSIS-DAP	LPC-Link2
LPCXpresso55S16	CMSIS-DAP	LPC-Link2
LPCXpresso55S28	CMSIS-DAP	LPC-Link2
LPCXpresso55S36	CMSIS-DAP	MCU-Link
LPCXpresso55S69	CMSIS-DAP	LPC-Link2
LPCXpresso802	CMSIS-DAP	LPC-Link2
LPCXpresso804	CMSIS-DAP	LPC-Link2
LPCXpresso824MAX	CMSIS-DAP	LPC-Link2
LPCXpresso845MAX	CMSIS-DAP	LPC-Link2
LPCXpresso860MAX	CMSIS-DAP	LPC-Link2

continues on next page

Table 1 – continued from previous page

Hardware platform	Default debugger firmware	On-board debugger probe
MC56F80000-EVK	P&E Micro	Multilink
MC56F81000-EVK	P&E Micro	Multilink
MC56F83000-EVK	P&E Micro	OSJTAG
MCIMX93-EVK	N/A	N/A
MCIMX93-QSB	N/A	N/A
MCIMX93AUTO-EVK	N/A	N/A
MCX-N5XX-EVK	CMSIS-DAP	MCU-Link
MCX-N9XX-EVK	CMSIS-DAP	MCU-Link
MCX-W71-EVK	CMSIS-DAP	MCU-Link
MCX-W72-EVK	CMSIS-DAP	MCU-Link
MIMXRT1024-EVK	CMSIS-DAP	LPC-Link2
MIMXRT1040-EVK	CMSIS-DAP	LPC-Link2
MIMXRT1060-EVKB	CMSIS-DAP	LPC-Link2
MIMXRT1060-EVKC	CMSIS-DAP	MCU-Link
MIMXRT1160-EVK	CMSIS-DAP	LPC-Link2
MIMXRT1170-EVKB	CMSIS-DAP	MCU-Link
MIMXRT1180-EVK	CMSIS-DAP	MCU-Link
MIMXRT685-AUD-EVK	CMSIS-DAP	LPC-Link2
MIMXRT700-EVK	CMSIS-DAP	MCU-Link
RD-RW612-BGA	CMSIS-DAP	MCU-Link
TWR-KM34Z50MV3	P&E Micro	OpenSDA v1
TWR-KM34Z75M	P&E Micro	OpenSDA v1
TWR-KM35Z75M	CMSIS-DAP	OpenSDA v2
TWR-MC56F8200	P&E Micro	OSJTAG
TWR-MC56F8400	P&E Micro	OSJTAG

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
}
```

Repository-Layout SDK Package

The Repository-Layout SDK Package is downloaded and extracted the same way as the Classic SDK Package — see SDK Builder for how to obtain it. Once extracted, it uses the same SDK

workspace layout as the GitHub Repository SDK, so you get the same offline development experience as a full workspace checkout. This section walks through installing development tools, touring the workspace layout, building your first example, generating IDE projects for IAR or Keil MDK, and importing the package into the MCUXpresso for VS Code extension.

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`

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.

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

Run a Demo Using IAR The Repository-Layout SDK Package does not ship pre-generated IDE projects. Instead, you generate an IAR Embedded Workbench project on demand from the CMake build, the same way you would from a GitHub Repository SDK checkout.

Prerequisite: Ruby

IDE project generation is implemented in Ruby and is only needed if you want to use IAR instead of the command line or VS Code. See *Ruby - IDE Project Generation (Optional)* for setup instructions before continuing.

1. Generate the IAR project

If this is a pristine build, specify board, example, toolchain and core on the command line:

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

If you already built the example with `west build`, you can regenerate the project with the shorter form:

```
west build -t guiproject
```

The project files are generated into `mcuxsdk/build/iar`, using relative paths to refer to source files and include paths in the repository. For the full set of `guiproject` options, see IDE Project Generation.

Generate into a custom folder

By default the project is generated into `mcuxsdk/build/<toolchain>`. To generate into a different location instead — for example to keep projects for several examples or boards side by side — pass `-d` with the pristine build command:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world --toolchain iar -Dcore_id=cm7 --config␣
↪flexspi_nor_debug -p always -t guiproject -d build_hello_world_iar
```

2. Open the project in IAR Embedded Workbench

Open the generated workspace or project file from:

```
mcuxsdk/build/iar
```

3. Build the demo application

1. Select the desired build target from the drop-down menu, for example **hello_world - flexspi_nor_debug**.
2. Click **Make** to build the demo application.
3. The build completes without errors.

4. Run the demo application

1. Open a terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (see *How to determine COM port*).
2. Click **Download and Debug** to flash and start debugging.
3. Click **Go** to run to the entry point and resume execution.
4. The terminal displays the demo application output.

Next steps

- *Run a demo using MDK* — the same flow for Keil MDK
- *Building Your First Project* — build and run from the command line

Run a Demo Using MDK The Repository-Layout SDK Package does not ship pre-generated IDE projects. Instead, you generate a Keil MDK (µVision) project on demand from the CMake build, the same way you would from a GitHub Repository SDK checkout.

Prerequisites

Ruby

IDE project generation is implemented in Ruby and is only needed if you want to use MDK instead of the command line or VS Code. See *Ruby - IDE Project Generation (Optional)* for setup instructions before continuing.

CMSIS device pack

Cortex Microcontroller Software Interface Standard (CMSIS) device packs must be installed in MDK to fully support the device from a debug perspective, including memory map information, register definitions, and flash programming algorithms. In µVision, select the **Pack Installer** icon and install the pack for your device before opening the generated project.

1. Generate the MDK project

If this is a pristine build, specify board, example, toolchain and core on the command line:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world --toolchain mdk -Dcore_id=cm7 --config␣
↪flexspi_nor_debug -p always -t guiproject
```

If you already built the example with `west build`, you can regenerate the project with the shorter form:

```
west build -t guiproject
```

The project files are generated into `mcuxsdk/build/mdk`, using relative paths to refer to source files and include paths in the repository. For the full set of `guiproject` options, see IDE Project Generation.

Generate into a custom folder

By default the project is generated into `mcuxsdk/build/<toolchain>`. To generate into a different location instead — for example to keep projects for several examples or boards side by side — pass `-d` with the pristine build command:

```
west build -b evkbmimxrt1170 examples/demo_apps/hello_world --toolchain mdk -Dcore_id=cm7 --config␣
↪flexspi_nor_debug -p always -t guiproject -d build_hello_world_mdk
```

2. Open the project in µVision

Open the generated `.uvmpw` workspace file from:

```
mcuxsdk/build/mdk
```

3. Build the demo application

1. Select **Rebuild** to build the demo project.
2. The build completes without errors.

4. Run the demo application

1. Open a terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (see *How to determine COM port*).
2. Click the **Download** button to download the application to the target.
3. Click the **Start/Stop Debug Session** button to enter the debug session.
4. Click the **Run** button to start the application.
5. The terminal displays the demo application output.

Next steps

- *Run a demo using IAR* — the same flow for IAR Embedded Workbench
- *Building Your First Project* — build and run from the command line

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.

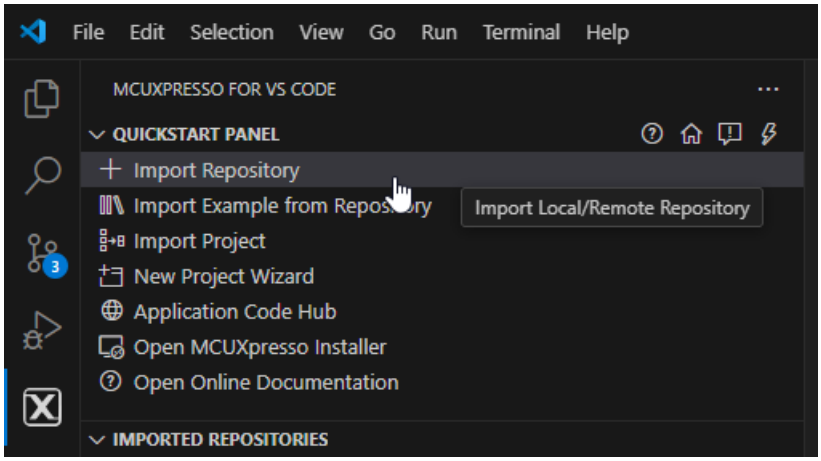
SDK Import and Setup

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.

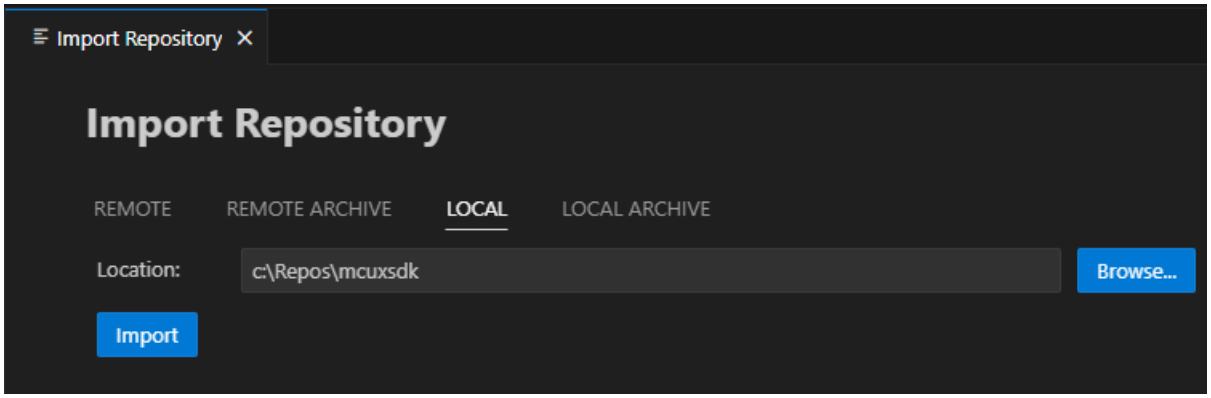
Import GitHub Repository SDK

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

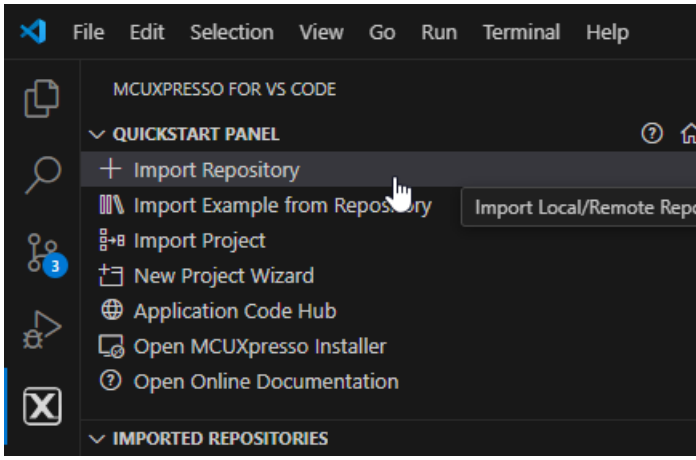


Note: You can import the SDK in several ways. Refer to the MCUXpresso for VS Code documentation for details.

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

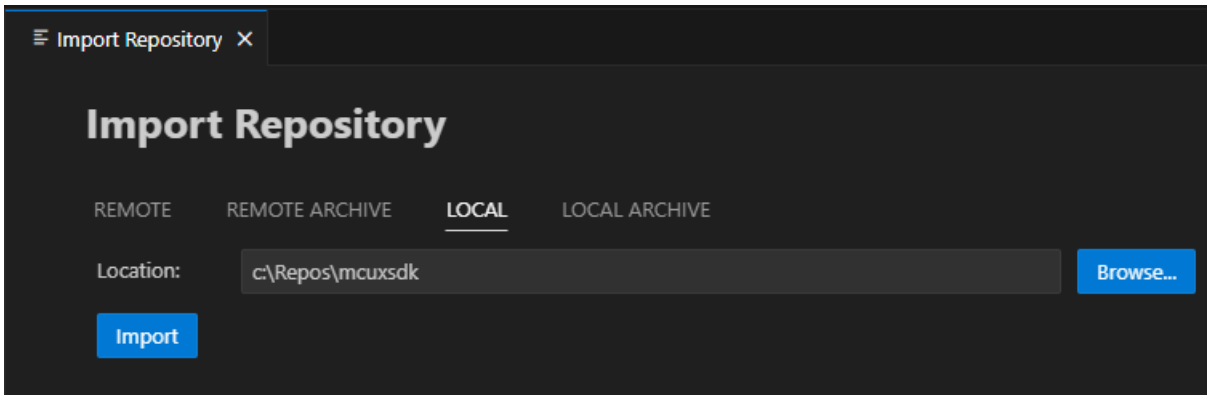


Import Repository-Layout SDK Package

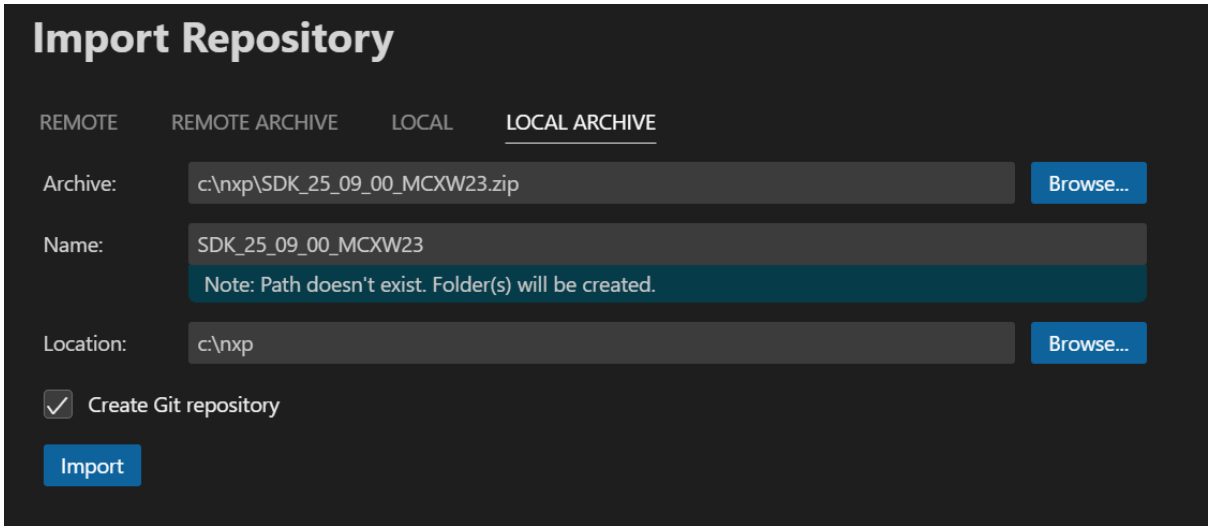


Click **Import Repository** from the **QUICKSTART PANEL**

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



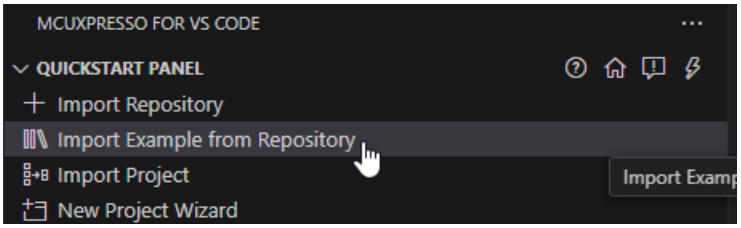
Else if the SDK is ZIP archive, select **Local Archive**, browse to the downloaded SDK ZIP file, fill the link of expect location, then click **Import**.



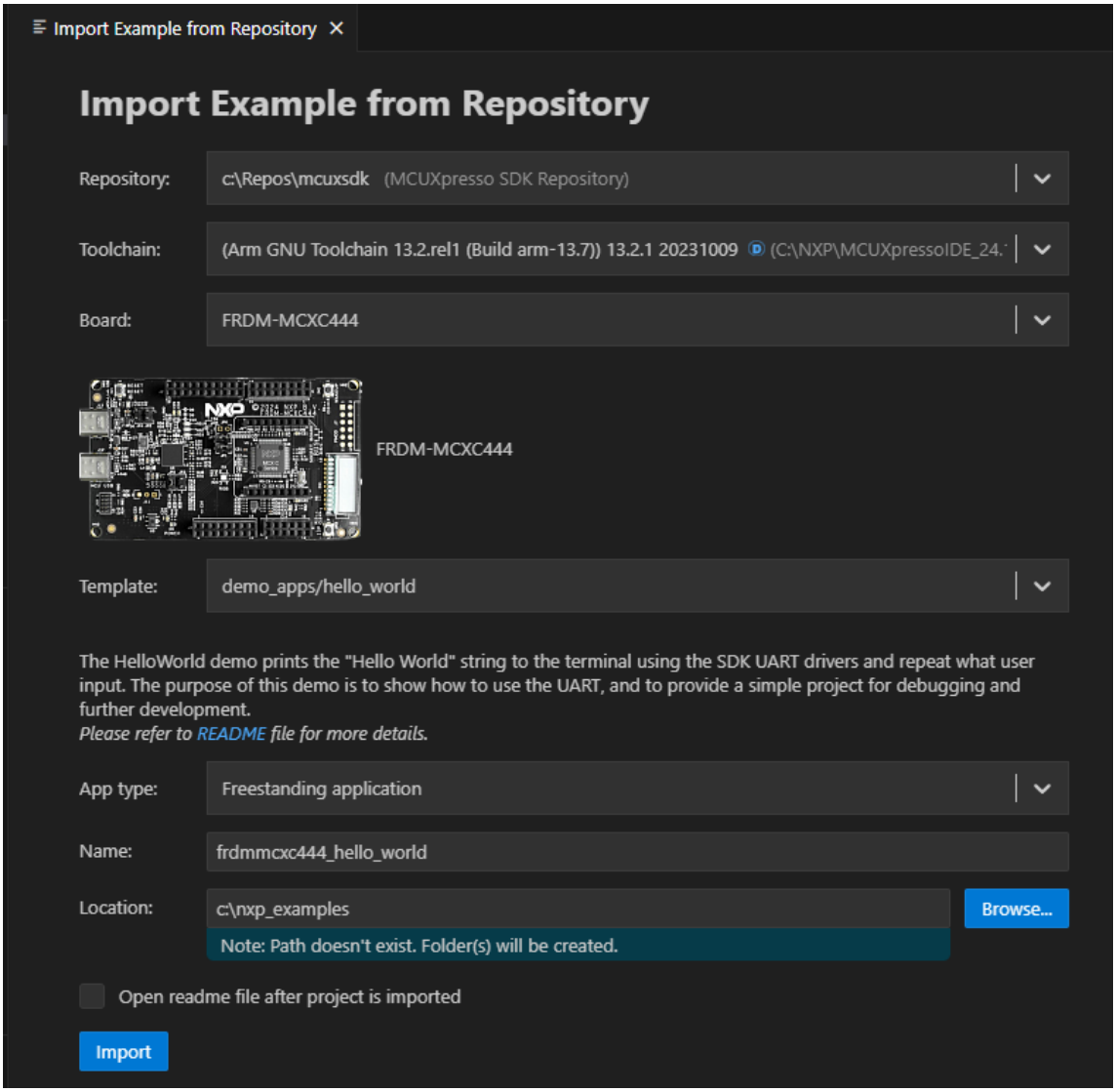
Building Example Applications

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

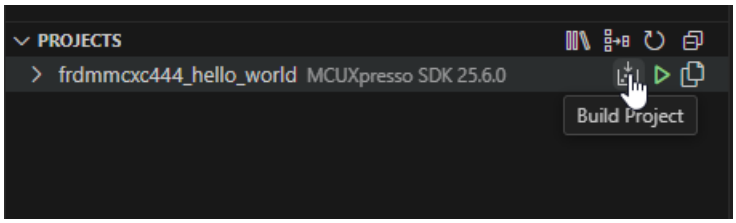
VS Code will prompt you to confirm if the imported files are trusted. Click **Yes** to proceed.

Building Projects

Build Process

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

3. Click the **Build Project** icon

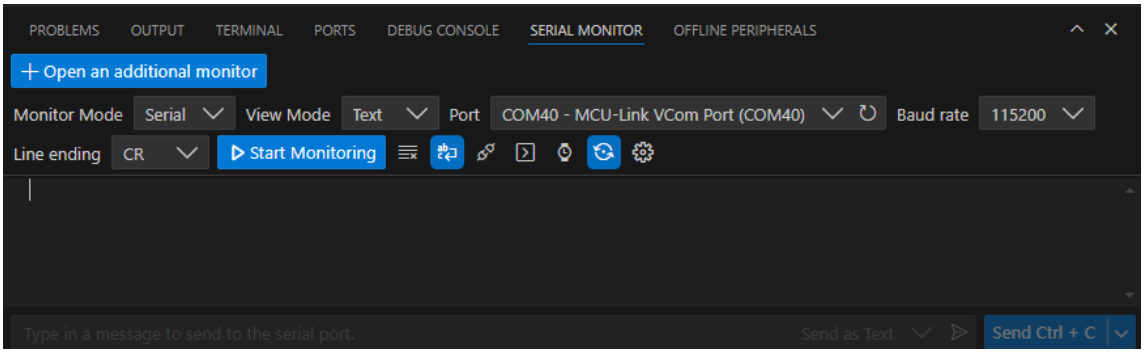


The integrated terminal will display build output at the bottom of the VS Code window.

Running and Debugging

Serial Monitor Setup

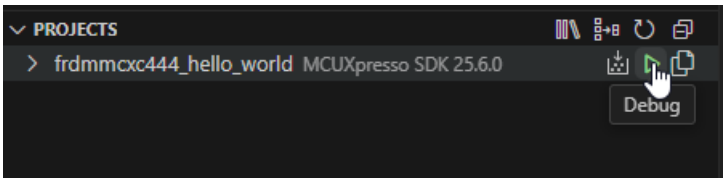
1. Open **Serial Monitor** from VS Code's integrated terminal



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



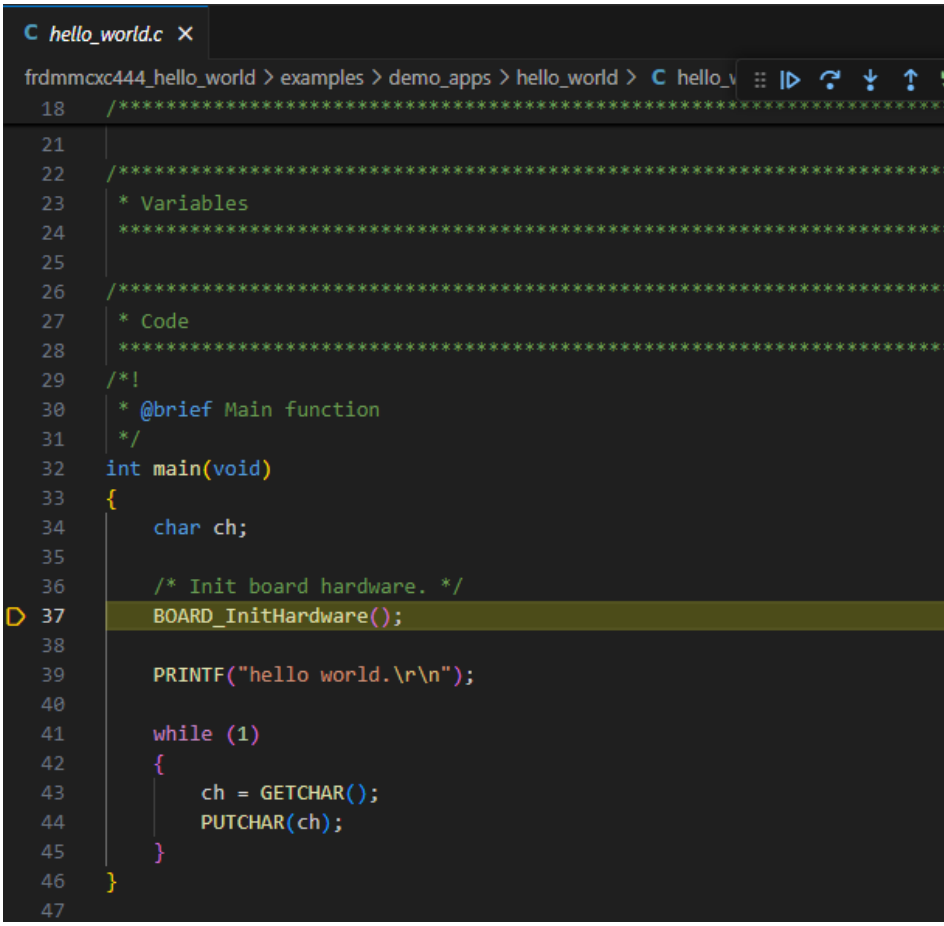
The debug session will begin with debug controls initially at the top of the interface.

Debug Controls

Use the debug controls to manage execution:

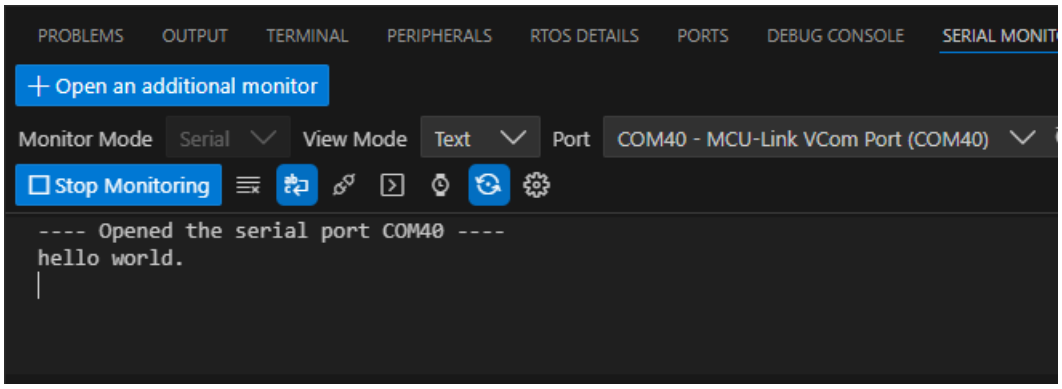
- **Continue**: Resume code execution
- **Step controls**: Navigate through code

- **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

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:

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.

Development Workflows

Get started with building and running projects:

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 Release Notes

1.5.1 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

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 board	MCU devices
FRDM K22F	MK22FN128VDC10, MK22FN128VLH10, MK22FN128VLL10, MK22FN128VMP10, MK22FN128CAH12, MK22FN256CAH12, MK22FN256VDC12, MK22FN256VLH12, MK22FN256VLL12, MK22FN256VMP12, MK22FN512CAP12, MK22FN512VDC12, MK22FN512VFX12, **MK22FN512VLH12**, MK22FN512VLL12, MK22FN512VMP12, MK02FN128VFM10, MK02FN128VLF10, MK02FN128VLH10, MK02FN64VFM10, MK02FN64VLF10, MK02FN64VLH10

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

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

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

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

LVGL LVGL Open Source Graphics Library

llhttp HTTP parser llhttp

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.

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.

Build warning in freertos_tickless example

A build warning appears in the freertos_tickless example while working in the ArmGCC environment.

```
.. code-block:: none
    `c:\c\pkg\cmsis\core\include\core_cm0plus.h:854:52: warning: array subscript 14 is above array bounds
↪of 'volatile uint32_t[8]' {aka 'volatile long unsigned int[8]'} [-Warray-bounds]`.
```

Cannot add SDK components into FreeRTOS projects

It is not possible to add any SDK components into FreeRTOS project using the MCUXpresso IDE New Project wizard.

Example freertos_lpspi fail before the console output

The example freertos_lpspi fails before the message “LPSPI master transfer completed successfully.” appears in the console output.

Console output:

```
```
FreeRTOS LPSPI example start.
This example use one lpspi instance as master and another as slave on a single board.
Master and slave are both use interrupt way.
Please make sure you make the correct line connection. Basically, the connection is:
LPSPI_master -- LPSPI_slave
 CLK -- CLK
 PCS -- PCS
 SOUT -- SIN
 SIN -- SOUT   ```
```

#### Importing FTM_PDB_ADC16 project into MCUXpresso Config tool reports FTM driver error

When you import the FTM_PDB_ADC16 project using the MCUXpresso Config Tool an error is prompted. The error prompts that the FTM driver is not aligned with the peripheral components. It is recommended not to import this project into the MCUXpresso Config Tool.

## 1.6 ChangeLog

### 1.6.1 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
```

ADC16

[2.3.0]

- Improvements
  - Added new API ADC16_EnableAsynchronousClockOutput() to enable/disable ADACK output.
  - In ADC16_GetDefaultConfig(), set enableAsynchronousClock to false.

[2.2.0]

- Improvements
  - Added hardware average mode in adc_config_t structure, then the hardware average mode can be set by invoking ADC16_Init() function.

[2.1.0]

- New Features:
  - Supported KM series' new ADC reference voltage source, bandgap from PMC.

[2.0.3]

- Bug Fixes
  - Fixed IAR warning Pa082: the order of volatile access should be defined.

[2.0.2]

- Improvements
  - Used conversion control feature macro instead of that in IO map.

[2.0.1]

- Bug Fixes
  - Fixed MISRA-2012 rules.
    - Rule 16.4, 10.1, 13.2, 14.4 and 17.7.

[2.0.0]

- Initial version

CLOCK

[2.5.3]

- Bug Fixes
  - Fixed an issue that in CLOCK_SetFbeMode() C4 register not updated.

[2.5.2]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 18.1.

[2.5.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, rule 10.8, rule 12.2 and so on.

[2.5.0]

- New Features
  - Moved SDK_DelayAtLeastUs function from clock driver to common driver.
  - Fixed IAR warning Pa082 for the clock driver.

[2.4.0]

- New Features
  - Added two APIs to set slow and fast internal reference clock variable.

[2.3.0]

- Bug Fixes
  - Fixed the issue for MISRA-2012 check.
    - Fixed rule 10.4, rule 10.1, rule 10.6, rule 13.5, rule 10.8.
  - Fixed incorrect External Oscillator Configuration sequence and ensure oscillator configuration be executed before it be enabled.
- New Features
  - Added new API CLOCK_DelayAtLeastUs() implemented by DWT to allow users set delay in unit of microsecond.

[2.2.1]

- Bug Fixes
  - Fixed the issue that MCG could not switch to FEE/FBE/PBE modes when OSCERCLK clock not enabled.

[2.2.0]

- New Features
  - Updated CLOCK_SetFeiMode/CLOCK_SetFbiMode/CLOCK_BootToFeiMode() to support set MCG_C4[DMX32]=1 in FEI/FBI modes.
- Bug Fixes
  - Updated IP_CLOCKS array, remove unused gates and add missing gates.

[2.1.0]

- Other Changes
  - Merged fsl_mcg and fsl_osc into fsl_clock.

[2.0.0]

- Initial version.

CMP

[2.0.3]

- Improvements
  - Updated to clear CMP settings in DeInit function.

[2.0.2]

- Bug Fixes
  - Fixed the violations of MISRA 2012 rules:
    - Rule 10.3

[2.0.1]

- Bug Fixes
  - Fixed MISRA-2012 rules.
    - Rule 14.4, rule 10.3, rule 10.1, rule 10.4 and rule 17.7.

[2.0.0]

- Initial version.

COMMON

[2.6.7]

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

[2.6.6]

- Improvements
  - Support Cortex M55.

[2.6.5]

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

[2.6.4]

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

[2.6.3]

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

[2.6.2]

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

[2.6.1]

- Improvements
  - Support Cortex M23.

[2.6.0]

- Bug Fixes
  - Fix CERT-C violations.

[2.5.0]

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

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.3]

- New Features
  - Added NETC into status group.

[2.3.2]

- Improvements
  - Make driver aarch64 compatible

[2.3.1]

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

[2.3.0]

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

[2.2.10]

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

[2.2.9]

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

[2.2.8]

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

[2.2.7]

- Other Change
  - Added MECC status group definition.

[2.2.6]

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

[2.2.5]

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

[2.2.4]

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

[2.2.3]

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

[2.2.2]

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

[2.2.1]

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

[2.2.0]

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

[2.1.4]

- New Features
  - Added OTFAD into status group.

[2.1.3]

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

[2.1.2]

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

[2.1.1]

- Bug Fixes
  - Deleted and optimized repeated macro.

[2.1.0]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

CRC

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

- Bug fix:
  - Fix MISRA issues.

[2.0.2]

- Bug fix:
  - Fix MISRA issues.

[2.0.1]

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

[2.0.0]

- Initial version.

DAC

[2.0.2]

- Bug Fixes
  - Fixed MISRA-2012 issues:
    - Rule 10.3, 10.8 and 17.7.

[2.0.1]

- Bug Fixes
  - Moved the default DAC_Enable(..., true) from DAC_Init() to the application code so that users can enable the DAC's output.

[2.0.0]

- Initial version.

DMAMUX

[2.1.4]

- Bug Fixes
  - Fixed CERT INT31-C violation in DMAMUX_SetSource by adding non-negative assertion and explicit cast.

[2.1.3]

- Improvements
  - Wrap DMAMUX_GetInstance into FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL to avoid build issues.

[2.1.2]

- Bug Fixes
  - Add macro FSL_DMAMUX_CHANNEL_NUM to calculat correct DMAMUX channel number when input EDAM channel number.

[2.1.1]

- Improvements
  - Add macro FSL_FEATURE_DMAMUX_CHANNEL_NEEDS_ENDIAN_CONVERT and DMAMUX_CHANNEL_ENDIAN_CONVERTn do channel endian convert.

[2.1.0]

- Improvements
  - Modify the type of parameter source from uint32_t to int32_t in the DMAMUX_SetSource.

[2.0.5]

- Improvements
  - Added feature FSL_FEATURE_DMAMUX_CHCFG_REGISTER_WIDTH for the difference of CHCFG register width.

[2.0.4]

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

[2.0.3]

- Bug Fixes
  - Fixed the issue for MISRA-2012 check.
    * Fixed rule 10.4 and rule 10.3.

[2.0.2]

- New Features
  - Added an always-on enable feature to a DMA channel for ULP1 DMAMUX support.

[2.0.1]

- Bug Fixes
  - Fixed the build warning issue by changing the type of parameter source from uint8_t to uint32_t when setting DMA request source in DMAMUX_SetSourceChange.

[2.0.0]

- Initial version.

DSPI

[2.2.8]

- New Features
  - Added common IRQ handler entry DSPI_DriverIRQHandler.

[2.2.7]

- Improvements
  - Added macros to disable some features.

[2.2.6]

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

[2.2.5]

- Improvements
  - Initialized the EDMA configuration structure in the DSPI EDMA driver.

[2.2.4]

- Bug Fixes
  - Fixed bug that instance with shared TR/RX EDMA request cannot transfer 1 datawidth of data as master in single transfer.

[2.2.3]

- Improvements
  - Added macro of getting maximum transfer size using EDMA way.

[2.2.2]

- Bug Fixes
  - MISRA C-2012 issue fixed:
    - Fixed rules, containing: 10.8, 10.3.
  - Fixed the build warning issue.
  - Fixed the bug that PCS would temporarily pull down for a while during master initialization.
  - Fixed compiling error of undefined identifier g_dspiDummyData.
  - Changed the type of kDSPI_AllStatusFlag and kDSPI_AllInterruptEnable from int to uint32_t to fix Pe068 warning of integer conversion.
  - Eliminated IAR Pa082 warning in DSPI_MasterTransferDMA.

[2.2.1]

- Bug Fixes
  - Fixed the bug for double execution of transfer complete callback in master interrupt transfer mode. In the interrupt routine, the DSPI interrupt may drop in the situation of the interrupt pending by itself while receiving the last frame, adding check to the transfer state to execute the callback function.
  - Fixed wrong logic in DSPI_SetFifoEnable().
  - MISRA C-2012 issue fixed.
    - Fixed rules, containing: rule-12.1, rule-17.7, rule-16.4, rule-14.4, rule-10.4, rule-10.8, rule-10.3, rule-10.1, rule-10.6, rule-13.5, rule-11.3, rule-13.2, rule-8.3, and rule-8.5.

[2.2.0]

- New Features
  - Added gasket feature for SPI EDMA driver, which reduces one channel used in the EDMA master transfer. With this feature support, only two channels are needed. For example, if the gasket feature is supported, we could use the DSPI_MasterTransferCreateHandleEDMA function like below: DSPI_MasterTransferCreateHandleEDMA(EXAMPLE_DSPI_MASTER_BASEADDR, &g_dspi_edma_m_handle, DSPI_MasterUserCallback, &userData, &dspiEdmaMasterRxRegToRxDataHandle, NULL, &dspiEdmaMasterIntermediaryToTxRegHandle);
  - 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 send and receive data by one API in the polling/interrupt/EDMA way, and they can choose either to transmit first or to receive first. Additionally, the PCS pin can be configured as assert status in transmission (between transmit and receive) by setting the isPcsAssertInTransfer to true.

[2.1.4]

- Bug Fixes
  - DSPI EDMA driver: the DSPI instance has been separated, so the DMA request source can now transfer up to 32767 Bytes data in one DSPI_MasterTransferEDMA() transfer.

[2.1.3]

- Bug Fixes
  - DSPI EDMA driver can no longer support the case that the transfer data size is odd, but the bitsPerFrame is greater than 8.
- Improvements
  - Added #ifndef/#endif to allow users to change the default TX value at compile time.

[2.1.2]

- Bug Fixes
  - DSPI_MasterTransferBlocking function would hang in some corner cases (for example, some cases with bitsPerFrame is 4,6 and kDSPI_MasterPcsContinuous transfer mode).

[2.1.1]

- Bug Fixes
  - Set the EOQ (End Of Queue) bit to TRUE for the last transfer in transactional APIs.

[2.1.0]

- New Features
  - Added Transfer prefix in transactional APIs.

[2.0.0]

- Initial version.

DSPI_EDMA

[2.2.7]

- Improvements
  - Added macro to disable slave functionality.

[2.2.6]

- Improvements
  - Add EDMA ext API to accommodate more types of EDMA.

[2.2.5]

- Refer DSPI driver change log 2.0.0 to 2.4.1

EDMA

[2.4.9]

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

[2.4.8]

- New Features
  - Added common IRQ handler entry EDMA_DriverIRQHandler.

[2.4.7]

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

[2.4.6]

- Bug Fixes
  - Fixed the EDMA header index retrieval error caused by done bit calculation mistake issue.

[2.4.5]

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

[2.4.4]

- Bug Fixes
  - Fixed comments by replacing STCD with TCD
  - Fixed the TCD overwrite issue when submit transfer request in the callback if there is a active TCD in hardware.
  - Fixed violations of MISRA C-2012 rule 10.8,5.6.

[2.4.3]

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

[2.4.2]

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

[2.4.1]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

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

[2.3.0]

- Improvements
  - Added new api EDMA_PrepareTransferConfig to allow different configurations of width and offset.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.4, 10.1.
  - Fixed the Coverity issue regarding out-of-bounds write.

[2.2.0]

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

[2.1.9]

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

[2.1.8]

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

[2.1.7]

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

[2.1.6]

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

[2.1.5]

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

[2.1.4]

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

[2.1.3]

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

[2.1.2]

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

[2.1.1]

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

[2.1.0]

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

[2.0.5]

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

[2.0.4]

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

[2.0.3]

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

[2.0.2]

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

[2.0.1]

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

[2.0.0]

- Initial version.

EWM

[2.0.4]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.0.3]

- Bug Fixes
  - Fixed violation of MISRA C-2012 rules: 10.1, 10.3.

[2.0.2]

- Bug Fixes
  - Fixed violation of MISRA C-2012 rules: 10.3, 10.4.

[2.0.1]

- Bug Fixes
  - Fixed the hard fault in EWM_Deinit.

[2.0.0]

- Initial version.

FLASH

[3.3.0]

- New Feature
  - Support for EEPROM Quick Write on devices with FTFC

[3.2.0]

- New Feature
  - Basic support for FTFC

[3.1.3]

- New Feature
  - Support 512KB flash for Kinetis E serials.

[3.1.2]

- Bug Fixes — Remove redundant comments.

[3.1.1]

- Bug Fixes — MISRA C-2012 issue fixed: rule 10.3

[3.1.0]

- New Feature
  - Support erase flash asynchronously.

[3.0.2]

- Bug Fixes — MISRA C-2012 issue fixed: rule 8.4, 17.7, 10.4, 16.1, 21.15, 11.3, 10.7 — building warning -Wnull-dereference on arm compiler v6

[3.0.1]

- New Features
  - Added support FlexNVM alias for (kw37/38/39).

[3.0.0]

- Improvements
  - Reorganized FTFx flash driver source file.
  - Extracted flash cache driver from FTFx driver.
  - Extracted flexnvm flash driver from FTFx driver.

[2.3.1]

- Bug Fixes
  - Unified Flash IFR design from K3.
  - New encoding rule for K3 flash size.

[2.3.0]

- New Features
  - Added support for device with LP flash (K3S/G).
  - Added flash prefetch speculation APIs.
- Improvements
  - Refined flash_cache_clear function.
  - Reorganized the member of flash_config_t struct.

[2.2.0]

- New Features
  - Supported FTFL device in FLASH_Swap API.
  - Supported various pflash start addresses.
  - Added support for KV58 in cache clear function.
  - Added support for device with secondary flash (KW40).
- Bug Fixes
  - Compiled execute-in-ram functions as PIC binary code for driver use.
  - Added missed flexram properties.
  - Fixed unaligned variable issue for execute-in-ram function code array.

[2.1.0]

- Improvements
  - Updated coding style to align with KSDK 2.0.
  - Different-alignment-size support for pflash and flexnvm.
  - Improved the implementation of execute-in-ram functions.

[2.0.0]

- Initial version

FLEXBUS

[2.1.1]

- Bug Fixes
- MISRA C-2012 issue fixed: rule 10.3, 10.4, and 14.4.

[2.1.0]

- Bug Fixes
  - Added secondary wait states (SWS), which enabled flag into configuration structure, and updated FLEXBUS_Init() function according to SWS.
  - Fixed MISRA issues regarding rule 10.1/10.4.

[2.0.2]

- Bug Fixes
  - Removed dissociated chip clearing operation while initializing the FLEXBUS module. Only the associated chip's register will be cleared.

[2.0.1]

- Bug Fixes
  - Corrected FLEXBUS_Deinit() function to disable clock.

[2.0.0]

- Initial version.

FTM

[2.7.6]

- Bug Fixes
  - Fixed QUADEN bit not cleared in FTM_SetupPwm() and FTM_SetupPwmMode() when switching from quadrature decoder mode to PWM mode. QUADEN has higher priority and overrides PWM operation if left set.
- Improvements
  - Added support for devices with reduced FTM register set by introducing new feature macros:
    - FSL_FEATURE_FTM_HAS_NO_COMBINE_COMBINE1: COMBINE register has no per-pair grouping.
    - FSL_FEATURE_FTM_HAS_NO_COMBINE_FAULTEN: COMBINE register has no FAULTEN field.
    - FSL_FEATURE_FTM_HAS_NO_MODE_FAULTM: MODE register has no FAULTM field.
    - FSL_FEATURE_FTM_HAS_NO_MODE_FAULTIE: MODE register has no FAULTIE field.
    - FSL_FEATURE_FTM_HAS_NO_FLTCTRL: FLTCTRL register is not present.

    - FSL_FEATURE_FTM_HAS_NO_FMS_FAULTF: FMS register has no FAULTF field.
  - Added assert checks to detect unsupported channel pair numbers on devices where COMBINE has no per-pair grouping (only pair 0 is valid).

[2.7.5]

- Bug Fixes
  - Corrected the bit shift calculation for fault input filter enable in FTM_SetupFaultInput().

[2.7.4]

- Bug Fixes
  - Fixed violations of the CERT INT31-C.
  - Fixed MISRA C-2012 issue: rule 10.1.

[2.7.3]

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

[2.7.2]

- Improvements
  - Add API FTM_ERRATA_010856 for ERR010856 workaround.

[2.7.1]

- Bug Fixes
  - Added function macro when accsee FLTCTRL register FSTATE bit to prevent access nonexistent register.
  - Added function macro to prevent access nonexistent FTM channel for API FTM_ConfigSinglePWM() and FTM_ConfigCombinePWM().

[2.7.0]

- Improvements
  - Support period dithering and edge dithering feature with new APIs:
    - FTM_SetPeriodDithering()
    - FTM_SetEdgeDithering()
  - Support get channel n output and input state feature with new APIs:
    - FTM_GetChannelOutputState()
    - FTM_GetChannelInputState()
  - Support configure deadtime for specific combined channel pair with new API:
    - FTM_SetPairDeadTime()
  - Support filter clock prescale, fault output state.

  - Support new APIs to configure PWM and Modified Combine PWM:
    * FTM_ConfigSinglePWM()
    * FTM_ConfigCombinePWM()
  - Support new API to configure channel software output control:
    * FTM_SetSoftwareOutputCtrl()
    * FTM_GetSoftwareOutputValue()
    * FTM_GetSoftwareOutputEnable()
  - Support new API to update FTM counter initial value, modulo value and chanle value:
    * FTM_SetInitialModuloValue()
    * FTM_SetChannelValue()

[2.6.1]

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

[2.6.0]

- Improvements
  - Added support to half and full cycle reload feature with new APIs:
    * FTM_SetLdok()
    * FTM_SetHalfCycPeriod()
    * FTM_LoadFreq()
- Bug Fixes
  - Set the HWRSTCNT and SWRSTCNT bits to optional at initialization.

[2.5.0]

- Improvements
  - Added FTM_CalculateCounterClkDiv to help calculates the counter clock prescaler.
  - Modify FTM_UpdatePwmDutycycle API to make it return pwm duty cycles status.
- Bug Fixes
  - Fixed TPM_SetupPwm can't configure 100% center align combined PWM issues.

[2.4.1]

- Bug Fixes
  - Added function macro to determine if FTM instance has only basic features, to prevent access to protected register bits.

[2.4.0]

- Improvements
  - Added CNTIN register initialization in FTM_SetTimerPeriod API.
  - Added a new API to read the captured value of a FTM channel configured in capture mode:
    - FTM_GetInputCaptureValue()

[2.3.0]

- Improvements
  - Added support of EdgeAligned/CenterAligned/Asymmetrical combine PWM mode in FTM_SetupPWM() and FTM_SetupPwmMode() APIs.
  - Remove kFTM_ComplementaryPwm from support PWM mode, and add new parameter “enableComplementary” in structure ftm_chnl_pwm_signal_param_t.
  - Rename FTM_SetupFault() API to FTM_SetupFaultInput() to avoid ambiguity.

[2.2.3]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 14.4 and 17.7.

[2.2.2]

- Bug Fixes
  - Fixed the issue that when FTM instance has only TPM features cannot be initialized by FTM_Init() function. By added function macro to assert FTM is TPM only instance.

[2.2.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.1, 10.3, 10.4, 10.6, 10.7 and 11.9.

[2.2.0]

- Bug Fixes
  - Fixed the issue of comparison between signed and unsigned integer expressions.
- Improvements
  - Added support of complementary mode in FTM_SetupPWM() and FTM_SetupPwmMode() APIs.
  - Added new parameter “enableDeadtime” in structure ftm_chnl_pwm_signal_param_t.

[2.1.1]

- Bug Fixes
  - Fixed COVERITY integer handing issue where the right operand of a left bit shift statement should not be a negative value. This appears in FTM_SetReloadPoints().

[2.1.0]

- Improvements
  - Added a new API FTM_SetupPwmMode() to allow the user to set the channel match value in units of timer ticks. New configure structure called ftm_chnl_pwm_config_param_t was added to configure the channel's PWM parameters. This API is similar with FTM_SetupPwm() API, but the new API will not set the timer period(MOD value), it will be useful for users to set the PWM parameters without changing the timer period.
- Bug Fixes
  - Added feature macro to enable/disable the external trigger source configuration.

[2.0.4]

- Improvements
  - Added a new API to enable DMA transfer:
    - FTM_EnableDmaTransfer()

[2.0.3]

- Bug Fixes
  - Updated the FTM driver to enable fault input after configuring polarity.

[2.0.2]

- Improvements
  - Added support to Quad Decoder feature with new APIs:
    - FTM_GetQuadDecoderFlags()
    - FTM_SetQuadDecoderModuloValue()
    - FTM_GetQuadDecoderCounterValue()
    - FTM_ClearQuadDecoderCounterValue()

[2.0.1]

- Bug Fixes
  - Updated the FTM driver to fix write to ELSA and ELSB bits.
  - FTM combine mode: set the COMBINE bit before writing to CnV register.

[2.0.0]

- Initial version.

GPIO

[2.8.6]

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

[2.8.5]

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

[2.8.4]

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

[2.8.3]

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

[2.8.2]

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

[2.8.1]

- Bug Fixes
  - Fixed CERT INT31-C issues.

[2.8.0]

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

[2.8.0]

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

[2.7.3]

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

[2.7.2]

- New Features
  - Support devices without PORT module.

[2.7.1]

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

[2.7.0]

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

[2.6.0]

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

[2.5.3]

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

[2.5.2]

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

[2.5.1]

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

[2.4.1]

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

[2.4.0]

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

[2.3.2]

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

[2.3.1]

- Improvements
  - Removed deprecated APIs.

[2.3.0]

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

[2.2.1]

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

[2.1.1]

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

[2.1.0]

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

I2C

[2.0.12]

- Bug Fixes
  - Set the C1[TXAK] before start of reading last byte to sent NACK in I2C_MasterReadBlocking().
  - Updated I2C_MasterTransferBlocking() to does not change xfer->subaddressSize.

[2.0.11]

- New Features
  - Added common IRQ handler entry I2C_DriverIRQHandler.

[2.0.10]

- Bug Fixes
  - Fixed coverity issues.

[2.0.9]

- Bug Fixes
  - Fixed the MISRA-2012 violations.
    - Fixed rule 8.4, 10.1, 10.4, 13.5, 20.8.

[2.0.8]

- Bug Fixes
  - Fixed the bug that DFEN bit of I2C Status register 2 could not be set in I2C_MasterInit.
  - MISRA C-2012 issue fixed: rule 14.2, 15.7, and 16.4.
  - Eliminated IAR Pa082 warnings from I2C_MasterTransferDMA and I2C_MasterTransferCallbackDMA by assigning volatile variables to local variables and using local variables instead.
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 11.9, 14.4, 15.7, 17.7.
- Improvements
  - Improved timeout mechanism when waiting certain state in transfer API.
  - Updated the I2C_WAIT_TIMEOUT macro to unified name I2C_RETRY_TIMES.
  - Moved the master manually acknowledge byte operation into static function I2C_MasterAckByte.
  - Fixed control/status clean flow issue inside I2C_MasterReadBlocking to avoid potential issue that pending status is cleaned before it's proceeded.

[2.0.7]

- Bug Fixes
  - Fixed the issue for MISRA-2012 check.
    - Fixed rule 11.9 ,15.7 ,14.4 ,10.4 ,10.8 ,10.3, 10.1, 10.6, 13.5, 11.3, 13.2, 17.7, 5.7, 8.3, 8.5, 11.1, 16.1.
  - Fixed Coverity issue of unchecked return value in I2C_RTOS_Transfer.
  - Fixed variable redefine issue by moving i2cBases from fsl_i2c.h to fsl_i2c.c.
- Improvements
  - Added I2C_MASTER_FACK_CONTROL macro to enable FACK control for master transfer receive flow with IP supporting double buffer, then master could hold the SCL by manually setting TX AK/NAK during data transfer.

[2.0.6]

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

[2.0.5]

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

[2.0.4]

- Bug Fixes
  - Added a proper handle for transfer config flag kI2C_TransferNoStartFlag to support transmit with kI2C_TransferNoStartFlag flag. Support write only or write+read with no start flag; does not support read only with no start flag.

[2.0.3]

- Bug Fixes
  - Removed enableHighDrive member in the master/slave configuration structure because the operation to HDRS bit is useless, the user need to use DSE bit in port register to configure the high drive capability.
  - Added register reset operation in I2C_MasterInit and I2C_SlaveInit APIs. Fixed issue where I2C could not switch between master and slave mode.
  - Improved slave IRQ handler to handle the corner case that stop flag and address match flag come synchronously.

[2.0.2]

- Bug Fixes
  - Fixed issue in master receive and slave transmit mode with no stop flag. The master could not succeed to start next transfer because the master could not send out re-start signal.

  - Fixed the out-of-order issue of data transfer due to memory barrier.
  - Added hold time configuration for slave. By leaving the SCL divider and MULT reset values when configured to slave mode, the setup and hold time of the slave is then reduced outside of spec for lower baudrates. This can cause intermittent arbitration loss on the master side.
- New Features
  - Added address nak event for master.
  - Added general call event for slave.

[2.0.1]

- New Features
  - Added double buffer enable configuration for SoCs which have the DFEN bit in S2 register.
  - Added flexible transmit/receive buffer size support in I2C_SlaveHandleIRQ.
  - Added start flag clear, address match, and release bus operation in I2C_SlaveWrite/ReadBlocking API.
- Bug Fixes
  - Changed the kI2C_SlaveRepeatedStartEvent to kI2C_SlaveStartEvent.

[2.0.0]

- Initial version.

LLWU

[2.0.5]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3.
  - Fixed the issue that function LLWU_SetExternalWakeupPinMode() does not work on 32-bit width platforms.

[2.0.4]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.3, 10.4, 10.6, 10.7, 11.3.
  - Fixed issue that LLWU_ClearExternalWakeupPinFlag may clear other filter flags by mistake on platforms with 32-bit LLWU registers.

[2.0.3]

- Bug Fixes
  - Fixed MISRA-2012 rules.
    * Rule 16.4.

[2.0.2]

- Improvements
  - Corrected driver function LLWU_SetResetPinMode parameter name.
- Bug Fixes
  - Fixed MISRA-2012 rules.
    - Rule 14.4, 10.8, 10.4, 10.3.

[2.0.1]

- Other Changes
  - Updates for KL8x.

[2.0.0]

- Initial version.

LPTMR

[2.2.1]

- Bug Fixes
  - Fix CERT INT31-C issues.

[2.2.0]

- Improvements
  - Updated lptmr_prescaler_clock_select_t, only define the valid options.

[2.1.1]

- Improvements
  - Updated the characters from "PTMR" to "LPTMR" in "FSL_FEATURE_PTMR_HAS_NO_PRESCALER_CLOCK_SOURCE_1_SUPPORT" feature definition.

[2.1.0]

- Improvements
  - Implement for some special devices' not supporting for all clock sources.
- Bug Fixes
  - Fixed issue when accessing CMR register.

[2.0.2]

- Bug Fixes
  - Fixed MISRA-2012 issues.
    - Rule 10.1.

[2.0.1]

- Improvements
  - Updated the LPTMR driver to support 32-bit CNR and CMR registers in some devices.

[2.0.0]

- Initial version.

LPUART

[2.11.0]

- New Feature
  - Added support to enable transmitter RTS and configure transmitter RTS polarity.
  - Added support to enable Transmit Data Inversion.
- Improvements
  - Updated to work on DSC platform.

[2.10.0]

- New Feature
  - Added support to configure RTS watermark.

[2.9.4]

- Improvements
  - Merged duplicate code.

[2.9.3]

- Improvements
  - Added timeout for while loops in LPUART_Deinit().

[2.9.2]

- Bug Fixes
  - Fixed coverity issues.

[2.9.1]

- Bug Fixes
  - Fixed coverity issues.

[2.9.0]

- New Feature
  - Added support for swap TXD and RXD pins.
  - Added common IRQ handler entry LPUART_DriverIRQHandler.

[2.8.3]

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

[2.8.2]

- Bug Fix
  - Fixed the bug that LPUART_TransferEnable16Bit controled by wrong feature macro.

[2.8.1]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - Fixed rule-5.3, rule-5.8, rule-10.4, rule-11.3, rule-11.8.

[2.8.0]

- Improvements
  - Added support of DATA register for 9bit or 10bit data transmit in write and read API. Such as: LPUART_WriteBlocking16bit, LPUART_ReadBlocking16bit, LPUART_TransferEnable16Bit LPUART_WriteNonBlocking16bit, LPUART_ReadNonBlocking16bit.

[2.7.7]

- Bug Fixes
  - Fixed the bug that baud rate calculation overflow when srcClock_Hz is 528MHz.

[2.7.6]

- Bug Fixes
  - Fixed LPUART_EnableInterrupts and LPUART_DisableInterrupts bug that blocks if the LPUART address doesn't support exclusive access.

[2.7.5]

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

[2.7.4]

- Improvements
  - Added support for atomic register accessing in LPUART_EnableInterrupts and LPUART_DisableInterrupts.

[2.7.3]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 15.7.

[2.7.2]

- Bug Fix
  - Fixed the bug that the OSR calculation error when lupart init and lpuart set baud rate.

[2.7.1]

- Improvements
  - Added support for LPUART_BASE_PTRS_NS in security mode in file fsl_lpuart.c.

[2.7.0]

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

[2.6.0]

- Bug Fixes
  - Fixed bug that when there are multiple lpuart instance, unable to support different ISR.

[2.5.3]

- Bug Fixes
  - Fixed comments by replacing unused status flags kLPUART_NoiseErrorInRxDataRegFlag and kLPUART_ParityErrorInRxDataRegFlag with kLPUART_NoiseErrorFlag and kLPUART_ParityErrorFlag.

[2.5.2]

- Bug Fixes
  - Fixed bug that when setting watermark for TX or RX FIFO, the value may exceed the maximum limit.
- Improvements
  - Added check in LPUART_TransferDMAHandleIRQ and LPUART_TransferEdmaHandleIRQ to ensure if user enables any interrupts other than transfer complete interrupt, the dma transfer is not terminated by mistake.

[2.5.1]

- Improvements
  - Use separate data for TX and RX in lpuart_transfer_t.
- Bug Fixes

  - Fixed bug that when ring buffer is used, if some data is received in ring buffer first before calling LPUART_TransferReceiveNonBlocking, the received data count returned by LPUART_TransferGetReceiveCount is wrong.

[2.5.0]

- Bug Fixes
  - Added missing interrupt enable masks kLPUART_Match1InterruptEnable and kLPUART_Match2InterruptEnable.
  - Fixed bug in LPUART_EnableInterrupts, LPUART_DisableInterrupts and LPUART_GetEnabledInterrupts that the BAUD[LBKDIE] bit field should be soc specific.
  - Fixed bug in LPUART_TransferHandleIRQ that idle line interrupt should be disabled when rx data size is zero.
  - Deleted unused status flags kLPUART_NoiseErrorInRxDataRegFlag and kLPUART_ParityErrorInRxDataRegFlag, since firstly their function are the same as kLPUART_NoiseErrorFlag and kLPUART_ParityErrorFlag, secondly to obtain them one data word must be read out thus interfering with the receiving process.
  - Fixed bug in LPUART_GetStatusFlags that the STAT[LBKDIF], STAT[MA1F] and STAT[MA2F] should be soc specific.
  - Fixed bug in LPUART_ClearStatusFlags that tx/rx FIFO is reset by mistake when clearing flags.
  - Fixed bug in LPUART_TransferHandleIRQ that while clearing idle line flag the other bits should be masked in case other status bits be cleared by accident.
  - Fixed bug of race condition during LPUART transfer using transactional APIs, by disabling and re-enabling the global interrupt before and after critical operations on interrupt enable register.
  - Fixed DMA/eDMA transfer blocking issue by enabling tx idle interrupt after DMA/eDMA transmission finishes.
- New Features
  - Added APIs LPUART_GetRxFifoCount/LPUART_GetTxFifoCount to get rx/tx FIFO data count.
  - Added APIs LPUART_SetRxFifoWatermark/LPUART_SetTxFifoWatermark to set rx/tx FIFO water mark.

[2.4.1]

- Bug Fixes
  - Fixed MISRA advisory 17.7 issues.

[2.4.0]

- New Features
  - Added APIs to configure 9-bit data mode, set slave address and send address.

[2.3.1]

- Bug Fixes
  - Fixed MISRA advisory 15.5 issues.

[2.3.0]

- Improvements
  - Modified LPUART_TransferHandleIRQ so that txState will be set to idle only when all data has been sent out to bus.
  - Modified LPUART_TransferGetSendCount so that this API returns the real byte count that LPUART has sent out rather than the software buffer status.
  - Added timeout mechanism when waiting for certain states in transfer driver.

[2.2.8]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - Fixed rule-10.3, rule-14.4, rule-15.5.
  - Eliminated Pa082 warnings by assigning volatile variables to local variables and using local variables instead.
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 10.8, 14.4, 11.6, 17.7.
- Improvements
  - Added check for kLPUART_TransmissionCompleteFlag in LPUART_WriteBlocking, LPUART_TransferHandleIRQ, LPUART_TransferSendDMACallback and LPUART_SendEDMACallback to ensure all the data would be sent out to bus.
  - Rounded up the calculated sbr value in LPUART_SetBaudRate and LPUART_Init to achieve more acurate baudrate setting. Changed osr from uint32_t to uint8_t since osr's bigest value is 31.
  - Modified LPUART_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.

[2.2.7]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - Fixed rule-12.1, rule-17.7, rule-14.4, rule-13.3, rule-14.4, rule-10.4, rule-10.8, rule-10.3, rule-10.7, rule-10.1, rule-11.6, rule-13.5, rule-11.3, rule-13.2, rule-8.3.

[2.2.6]

- Bug Fixes
  - Fixed the issue of register's being in repeated reading status while dealing with the IRQ routine.

[2.2.5]

- Bug Fixes
  - Do not set or clear the TIE/RIE bits when using LPUART_EnableTxDMA and LPUART_EnableRxDMA.

[2.2.4]

- Improvements
  - Added hardware flow control function support.
  - Added idle-line-detecting feature in LPUART_TransferNonBlocking function. If an idle line is detected, a callback is triggered with status kStatus_LPUART_IdleLineDetected returned. This feature may be useful when the received Bytes is less than the expected received data size. Before triggering the callback, data in the FIFO (if has FIFO) is read out, and no interrupt will be disabled, except for that the receive data size reaches 0.
  - Enabled the RX FIFO watermark function. With the idle-line-detecting feature enabled, users can set the watermark value to whatever you want (should be less than the RX FIFO size). Data is received and a callback will be triggered when data receive ends.

[2.2.3]

- Improvements
  - Changed parameter type in LPUART_RTOS_Init struct from rtos_lpuart_config to lpuart_rtos_config_t.
- Bug Fixes
  - Disabled LPUART receive interrupt instead of all NVICs when reading data from ring buffer. Otherwise when the ring buffer is used, receive nonblocking method will disable all NVICs to protect the ring buffer. This may has a negative effect on other IPs that are using the interrupt.

[2.2.2]

- Improvements
  - Added software reset feature support.
  - Added software reset API in LPUART_Init.

[2.2.1]

- Improvements
  - Added separate RX/TX IRQ number support.

[2.2.0]

- Improvements
  - Added support of 7 data bits and MSB.

[2.1.1]

- Improvements
  - Removed unnecessary check of event flags and assert in LPUART_RTOS_Receive.
  - Added code to always wait for RX event flag in LPUART_RTOS_Receive.

[2.1.0]

- Improvements
  - Update transactional APIs.

LPUART_EDMA

[2.4.0]

- Refer LPUART driver change log 2.1.0 to 2.4.0

MCM

[2.2.1]

- Improvements
  - Add feature to support platforms that do not have the MCM CPCR (Core Platform Control Register).

[2.2.0]

- Improvements
  - Support platforms with less features.

[2.1.0]

- Others
  - Remove byteID from mcm_lmem_fault_attribute_t for document update.

[2.0.0]

- Initial version.

PDB

[2.0.4]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1 and 10.4.

[2.0.3]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 17.7.

[2.0.2]

- Improvement:
  - Used macros in feature file instead of that in IO map.

[2.0.1]

- Changed PDB register base array to const.

[2.0.0]

- Initial version.

PIT

[2.2.0]

- Bug Fixes
  - According to ERR050763, PIT_LDVAL_STAT register is not reliable in dynamic load mode, so remove the status check in PIT_SetRtiTimerPeriod which added since 2.1.1.
  - Removed not used bit PIT_RTI_TCTRL_CHN_MASK.
- Improvements
  - Added more guide about get RTI load status in PIT_SetRtiTimerPeriod's API comment.
  - Change PIT_RTI_Deinit to inline API.
  - Ensure PIT peripheral clock enabled in PIT_RTI_Init.
- New Features
  - Added PIT_ClearRtiSyncStatus API to clear the RTI_LDVAL_STAT register.

[2.1.1]

- Bug Fixes
  - Enable PIT when using RTI to ensure RTI can work properly in debug mode.
- Improvements
  - Added status check in PIT_SetRtiTimerPeriod to ensure the load value is synchronized into the RTI clock domain.
  - Added note for PIT_RTI_Init to remind users wait RTI sync.

[2.1.0]

- New Features
  - Support RTI (Real Time Interrupt) timer.

[2.0.5]

- Improvements
  - Support workaround for ERR007914. This workaround guarantee the write to MCR register is not ignored.

[2.0.4]

- Bug Fixes
  - Fixed PIT_SetTimerPeriod implementation, the load value trigger should be PIT clock cycles minus 1.

[2.0.3]

- Bug Fixes
  - Clear all status bits for all channels to make sure the status of all TCTRL registers is clean.

[2.0.2]

- Bug Fixes
  - Fixed MISRA-2012 issues.
    - Rule 10.1.

[2.0.1]

- Bug Fixes
  - Cleared timer enable bit for all channels in function PIT_Init() to make sure all channels stay in disable status before setting other configurations.
  - Fixed MISRA-2012 rules.
    - Rule 14.4, rule 10.4.

[2.0.0]

- Initial version.

PMC

[2.0.4]

- Bug Fixes
  - Add validation before narrowing uint32_t to uint8_t conversions using assert
  - Replace direct boolean to integer casts with explicit ternary operators
  - Add INT31-C compliance comments for safe narrowing conversions
  - Apply fixes to PMC_ConfigureLowVoltDetect, PMC_ConfigureLowVoltWarning, and PMC_ConfigureBandgapBuffer functions

[2.0.3]

- Bug Fixes
  - Fixed the violation of MISRA C-2012 rule 11.3.

[2.0.2]

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

[2.0.1]

- Bug Fixes
  - Fixed MISRA issues.
    - Rule 10.8, Rule 10.3.

[2.0.0]

- Initial version.

PORT

[2.5.1]

- Bug Fixes
  - Fix CERT INT31-C issues.
  - Fixed the violations of MISRA C-2012 rules: 10.1.

[2.5.0]

- Bug Fixes
  - Correct the kPORT_MuxAsGpio for some platforms.

[2.4.1]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules: 10.1, 10.8 and 14.4.

[2.4.0]

- New Features
  - Updated port_pin_config_t to support input buffer and input invert.

[2.3.0]

- New Features
  - Added new APIs for Electrical Fast Transient(EFT) detect.
  - Added new API to configure port voltage range.

[2.2.0]

- New Features
  - Added new api PORT_EnablePinDoubleDriveStrength.

[2.1.1]

- Bug Fixes
  - Fixed the violations of MISRA C-2012 rules: 10.1, 10.4□11.3□11.8, 14.4.

[2.1.0]

- New Features
  - Updated the driver code to adapt the case of the interrupt configurations in GPIO module. Will move the pin configuration APIs to GPIO module.

[2.0.2]

- Other Changes
  - Added feature guard macros in the driver.

[2.0.1]

- Other Changes
  - Added “const” in function parameter.
  - Updated some enumeration variables’ names.

RCM

[2.0.6]

- Bug Fixes
  - Fixed RCM_GetVersionId() always returning 0 for the major version field due to a wrong mask/shift combination (CID 8644199).

[2.0.5]

- Bug Fixes
  - Replace direct boolean to integer casts with explicit ternary operators for INT31-C compliance.
  - Add validation before narrowing uint32_t to uint8_t conversion using assert.
  - Add INT31-C compliance comments for safe type handling.
  - Apply fixes to RCM_ConfigureResetPinFilter and RCM_SetForceBootRomSource functions.

[2.0.4]

- Bug Fixes
  - Fixed violation of MISRA C-2012 rule 10.3

[2.0.3]

- Bug Fixes
  - Fixed violation of MISRA C-2012 rules.

[2.0.2]

- Bug Fixes
  - Fixed MISRA issue.
    - Rule 10.8, rule 10.1, rule 13.2, rule 3.1.

[2.0.1]

- Bug Fixes
  - Fixed kRCM_SourceSw bit shift issue.

[2.0.0]

- Initial version.

RNGA

[2.0.2]

- Bug fix:
  - Fix MISRA issue.

[2.0.1]

- Bug fix:
  - Fixed C++ build warning in RNGA driver.

[2.0.0]

- Initial version.

RTC

[2.4.0]

- New features
  - Add support for RTC clock output.
  - Add support for RTC time seconds interrupt configuration.

[2.3.3]

- Bug Fixes
  - Fix RTC_GetDatetime function validating datetime issue.

[2.3.2]

- Improvements
  - Handle errata 010716: Disable the counter before setting alarm register and then reenable the counter.

[2.3.1]

- Bug Fixes
  - Fixed CERT INT31-C violations.

[2.3.0]

- Improvements
  - Added API RTC_EnableLPOClock to set 1kHz LPO clock.
  - Added API RTC_EnableCrystalClock to replace API RTC_SetClockSource.

[2.2.2]

- Improvements
  - Refine _rtc_interrupt_enable order.

[2.2.1]

- Bug Fixes
  - Fixed the issue of Pa082 warning.
  - Fixed the issue of bit field mask checking.
  - Fixed the issue of hard code in RTC_Init.

[2.2.0]

- Bug Fixes
  - Fixed MISRA C-2012 issue.
    - Fixed rule contain: rule-17.7, rule-14.4, rule-10.4, rule-10.7, rule-10.1, rule-10.3.
  - Fixed central repository code formatting issue.
- Improvements
  - Added an API for enabling wakeup pin.

[2.1.0]

- Improvements
  - Added feature macro check for many features.

[2.0.0]

- Initial version.

SAI

[2.4.11]

- Bug Fixes
  - Fixed MSG findings for platforms with multiple FIFO size support.

[2.4.10]

- Improvements
  - Allow enabling/disabling implicit channel configuration.
  - Allow NULL FIFO watermark.
- Bug Fixes
  - Fix compilation warnings when asserts are disabled

[2.4.9]

- Added Errata ERR051421 workaround.

[2.4.8]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

[2.4.7]

- Added conditional support for bit clock swap feature
- Added common IRQ handler entry SAI_DriverIRQHandler.

[2.4.6]

- Bug Fixes
  - Fixed the IAR build warning.

[2.4.5]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

[2.4.4]

- Bug Fixes
  - Fixed enumeration sai_fifo_combine_t - add RX configuration.

[2.4.3]

- Bug Fixes
  - Fixed enumeration sai_fifo_combine_t value configuration issue.

[2.4.2]

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

[2.4.1]

- Bug Fixes
  - Fixed bitWidth incorrectly assigned issue.

[2.4.0]

- Improvements
  - Removed deprecated APIs.

[2.3.8]

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

[2.3.7]

- Improvements
  - Change feature “FSL_FEATURE_SAI_FIFO_COUNT” to “FSL_FEATURE_SAI_HAS_FIFO”.
  - Added feature “FSL_FEATURE_SAI_FIFO_COUNTn(x)” to align SAI fifo count function with IP in function

[2.3.6]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 5.6.

[2.3.5]

- Improvements
  - Make driver to be aarch64 compatible.

[2.3.4]

- Bug Fixes
  - Corrected the fifo combine feature macro used in driver.

[2.3.3]

- Bug Fixes
  - Added bit clock polarity configuration when sai act as slave.
  - Fixed out of bound access coverity issue.
  - Fixed violations of MISRA C-2012 rule 10.3, 10.4.

[2.3.2]

- Bug Fixes
  - Corrected the frame sync configuration when sai act as slave.

[2.3.1]

- Bug Fixes
  - Corrected the peripheral name in function SAI0_DriverIRQHandler.
  - Fixed violations of MISRA C-2012 rule 17.7.

[2.3.0]

- Bug Fixes
  - Fixed the build error caused by the SOC has no fifo feature.

[2.2.3]

- Bug Fixes
  - Corrected the peripheral name in function SAI0_DriverIRQHandler.

[2.2.2]

- Bug Fixes
  - Fixed the issue of MISRA 2004 rule 9.3.
  - Fixed sign-compare warning.
  - Fixed the PA082 build warning.
  - Fixed sign-compare warning.
  - Fixed violations of MISRA C-2012 rule 10.3,17.7,10.4,8.4,10.7,10.8,14.4,17.7,11.6,10.1,10.6,8.4,14.3,16.4,18
  - Allow to reset Rx or Tx FIFO pointers only when Rx or Tx is disabled.
- Improvements
  - Added 24bit raw audio data width support in sai sdma driver.
  - Disabled the interrupt/DMA request in the SAI_Init to avoid generates unexpected sai FIFO requests.

[2.2.1]

- Improvements
  - Added mclk post divider support in function SAI_SetMasterClockDivider.
  - Removed useless configuration code in SAI_RxSetSerialDataConfig.
- Bug Fixes
  - Fixed the SAI SDMA driver build issue caused by the wrong structure member name used in the function SAI_TransferRxSetConfigSDMA/SAI_TransferTxSetConfigSDMA.
  - Fixed BAD BIT SHIFT OPERATION issue caused by the FSL_FEATURE_SAI_CHANNEL_COUNTn.
  - Applied ERR05144: not set FCONT = 1 when TMR > 0, otherwise the TX may not work.

[2.2.0]

- Improvements
  - Added new APIs for parameters collection and simplified user interfaces:
    - SAI_Init
    - SAI_SetMasterClockConfig
    - SAI_TxSetBitClockRate
    - SAI_TxSetSerialDataConfig
    - SAI_TxSetFrameSyncConfig
    - SAI_TxSetFifoConfig
    - SAI_TxSetBitclockConfig
    - SAI_TxSetConfig
    - SAI_TxSetTransferConfig
    - SAI_RxSetBitClockRate
    - SAI_RxSetSerialDataConfig
    - SAI_RxSetFrameSyncConfig
    - SAI_RxSetFifoConfig
    - SAI_RxSetBitclockConfig
    - SAI_RXSetConfig
    - SAI_RxSetTransferConfig
    - SAI_GetClassicI2SConfig
    - SAI_GetLeftJustifiedConfig
    - SAI_GetRightJustifiedConfig
    - SAI_GetTDMConfig

[2.1.9]

- Improvements
  - Improved SAI driver comment for clock polarity.
  - Added enumeration for SAI for sample inputs on different edges.
  - Changed FSL_FEATURE_SAI_CHANNEL_COUNT to FSL_FEATURE_SAI_CHANNEL_COUNTn(base) for the difference between the different SAI instances.
- Added new APIs:
  - SAI_TxSetBitClockDirection
  - SAI_RxSetBitClockDirection
  - SAI_RxSetFrameSyncDirection
  - SAI_TxSetFrameSyncDirection

[2.1.8]

- Improvements
  - Added feature macro test for the sync mode2 and mode 3.
  - Added feature macro test for masterClockHz in sai_transfer_format_t.

[2.1.7]

- Improvements
  - Added feature macro test for the mclkSource member in sai_config_t.
  - Changed "FSL_FEATURE_SAI5_SAI6_SHARE_IRQ" to "FSL_FEATURE_SAI_SAI5_SAI6_SHARE_IRQ".
  - Added #ifndef #endif check for SAI_XFER_QUEUE_SIZE to allow redefinition.
- Bug Fixes
  - Fixed build error caused by feature macro test for mclkSource.

[2.1.6]

- Improvements
  - Added feature macro test for mclkSourceClockHz check.
  - Added bit clock source name for general devices.
- Bug Fixes
  - Fixed incorrect channel numbers setting while calling RX/TX set format together.

[2.1.5]

- Bug Fixes
  - Corrected SAI3 driver IRQ handler name.
  - Added I2S4/5/6 IRQ handler.
  - Added base in handler structure to support different instances sharing one IRQ number.
- New Features
  - Updated SAI driver for MCR bit MICS.
  - Added 192 KHZ/384 KHZ in the sample rate enumeration.
  - Added multi FIFO interrupt/SDMA transfer support for TX/RX.
  - Added an API to read/write multi FIFO data in a blocking method.
  - Added bclk bypass support when bclk is same with mclk.

[2.1.4]

- New Features
  - Added an API to enable/disable auto FIFO error recovery in platforms that support this feature.
  - Added an API to set data packing feature in platforms which support this feature.

[2.1.3]

- New Features
  - Added feature to make I2S frame sync length configurable according to bitWidth.

[2.1.2]

- Bug Fixes
  - Added 24-bit support for SAI eDMA transfer. All data shall be 32 bits for send/receive, as eDMA cannot directly handle 3-Byte transfer.

[2.1.1]

- Improvements
  - Reduced code size while not using transactional API.

[2.1.0]

- Improvements
  - API name changes:
    - SAI_GetSendRemainingBytes -> SAI_GetSentCount.
    - SAI_GetReceiveRemainingBytes -> SAI_GetReceivedCount.
    - All names of transactional APIs were added with “Transfer” prefix.
    - All transactional APIs use base and handle as input parameter.
    - Unified the parameter names.
- Bug Fixes
  - Fixed WLC bug while reading TCSR/RCSR registers.
  - Fixed MOE enable flow issue. Moved MOE enable after MICS settings in SAI_TxInit/SAI_RxInit.

[2.0.0]

- Initial version.

SAI_EDMA

[2.7.4]

- Bug Fixes
  - Fixed MSG finding for platforms with multiple FIFO size support.

[2.7.3]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.1, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 12.4.

[2.7.2]

- Improvements
  - Add macros MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL and MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL to let the user decide whether to enable SAI when calling SAI_TransferSendEDMA/SAI_TransferReceiveEDMA.

[2.7.1]

- Improvements
  - Add EDMA ext API to accommodate more types of EDMA.

[2.7.0]

- Improvements
  - Updated api SAI_TransferReceiveEDMA to support voice channel block interleave transfer.
  - Updated api SAI_TransferSendEDMA to support voice channel block interleave transfer.
  - Added new api SAI_TransferSetInterleaveType to support channel interleave type configurations.

[2.6.0]

- Improvements
  - Removed deprecated APIs.

[2.5.1]

- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 20.7.

[2.5.0]

- Improvements
  - Added new api SAI_TransferSendLoopEDMA/SAI_TransferReceiveLoopEDMA to support loop transfer.
  - Added multi sai channel transfer support.

[2.4.0]

- Improvements
  - Added new api SAI_TransferGetValidTransferSlotsEDMA which can be used to get valid transfer slot count in the sai edma transfer queue.
  - Deprecated the api SAI_TransferRxSetFormatEDMA and SAI_TransferTxSetFormatEDMA.
- Bug Fixes
  - Fixed violations of MISRA C-2012 rule 10.3,10.4.

[2.3.2]

- Refer SAI driver change log 2.1.0 to 2.3.2

SIM

[2.2.0]

- Improvements
  - Added API to trigger TRGMUX.

[2.1.3]

- Improvements
  - Updated function SIM_GetUniqueId to support different register names.

[2.1.2]

- Bug Fixes
  - Fixed SIM_GetUniqueId bug that could not get UIDH.

[2.1.1]

- Bug Fixes
  - Fixed violations of the MISRA C-2012 rules 10.1, 10.4

[2.1.0]

- Improvements
  - Added new APIs: SIM_GetRfAddr() and SIM_EnableSystickClock().

[2.0.0]

- Initial version.

SMC

[2.0.9]

- Bug Fixes
  - Replace width-unsafe register mask inversion with a $\mathrm{smc_reg_t}$-aware helper for INT31-C compliance

[2.0.8]

- Bug Fixes
  - Replace direct boolean to integer casts with explicit ternary operators for INT31-C compliance
  - Add validation before narrowing uint32_t to uint8_t conversion using assert
  - Add INT31-C compliance comments for safe type handling

  - Apply fixes to RCM_ConfigureResetPinFilter and RCM_SetForceBootRomSource functions

[2.0.7]

- Bug Fixes
  - Fixed MISRA-2012 issue 10.3.

[2.0.6]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - Fixed rule 10.3, rule 11.3.

[2.0.5]

- Bug Fixes
  - Fixed issue for MISRA-2012 check.
    - Fixed rule 15.7, rule 14.4, rule 10.3, rule 10.1, rule 10.4.

[2.0.4]

- Bug Fixes
  - When entering stop modes, used RAM function for the flash synchronization issue. Application should make sure that, the RW data of fsl_smc.c is located in memory region which is not powered off in stop modes.

[2.0.3]

- Improvements
  - Added APIs SMC_PreEnterStopModes, SMC_PreEnterWaitModes, SMC_PostExitWaitModes, and SMC_PostExitStopModes.

[2.0.2]

- Bug Fixes
  - Added DSB before WFI while ISB after WFI.
- Other Changes
  - Updated SMC_SetPowerModeVlpw implementation.

[2.0.1]

- Other Changes
  - Updated for KL8x.

[2.0.0]

- Initial version.

UART

[2.5.2]

- New Features
  - Added common IRQ handler entry UART_DriverIRQHandler.

[2.5.1]

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

[2.5.0]

- New Features
  - Added APIs UART_GetRxFifoCount/UART_GetTxFifoCount to get rx/tx FIFO data count.
  - Added APIs UART_SetRxFifoWatermark/UART_SetTxFifoWatermark to set rx/tx FIFO water mark.
- Bug Fixes
  - Fixed bug of race condition during UART transfer using transactional APIs, by disabling and re-enabling the global interrupt before and after critical operations on interrupt enable registers.
  - Fixed DMA/eDMA transfer blocking issue by enabling tx idle interrupt after DMA/eDMA transmission finishes.

[2.4.0]

- New Features
  - Added APIs to configure 9-bit data mode, set slave address and send address.

[2.3.0]

- Bug Fixes
  - Fixed the bug that, when framing/parity/noise/overflow flag or idle line detect flag is set, receive FIFO should be flushed to avoid FIFO pointer being in unknown state, since FIFO has no valid data.
- Improvements
  - Modified UART_TransferHandleIRQ so that txState will be set to idle only when all data has been sent out to bus.
  - Modified UART_TransferGetSendCount so that this API returns the real byte count that UART has sent out rather than the software buffer status.
  - Added timeout mechanism when waiting for certain states in transfer driver.

[2.2.0]

- New Features
  - Added UART hardware FIFO enable/disable API.
- Improvements
  - Added check for kUART_TransmissionCompleteFlag in UART_TransferHandleIRQ, UART_SendEDMACallback and UART_TransferSendDMACallback to ensure all the data would be sent out to bus.
- Bug Fixes
  - Eliminated IAR Pa082 warnings from UART_TransferGetRxRingBufferLength, UART_GetEnabledInterrupts, UART_GetStatusFlags and UART_TransferHandleIRQ.
  - Added code in UART_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.
  - Fixed MISRA issues.
    - Fixed rules 10.1, 10.3, 10.4, 14.4, 11.6, 17.7.

[2.1.6]

- Bug Fixes
  - Fixed the issue of register's being in repeatedly reading status while performing the IRQ routine.

[2.1.5]

- Improvements
  - Added hardware flow control function support.
  - Added idle-line-detecting feature in UART_TransferNonBlocking function. If an idle line is detected, a callback will be triggered with status kStatus_UART_IdleLineDetected returned. This feature may be useful when the number of received bytes is less than the expected receive data size. Before triggering the callback, data in the FIFO is read out (if it has FIFO), and no interrupt will be disabled except for the case that the receive data size reaches 0.
  - Enabled the RX FIFO watermark function. With the idle-line-detecting feature enabled, you can set the watermark value to whatever you want (should not be bigger than the RX FIFO size). Data is then received and a callback will be triggered when data receive ends.

[2.1.4]

- Improvements
  - Changed parameter type in UART_RTOS_Init() struct rtos_uart_config –> uart_rtos_config_t.
- Bug Fixes
  - Disabled UART receive interrupt instead of global interrupt when reading data from ring buffer. With ring buffer used, receive nonblocking will disable global interrupt to protect the ring buffer. This has a negative effect on other IPs using interrupt.

[2.1.3]

- New Features
  - Added RX framing error and parity error status check when using interrupt transfer.

[2.1.2]

- Bug Fixes
  - Fixed baud rate fine adjust bug to make the computed baud rate more accurate.

[2.1.1]

- Bug Fixes
  - Removed needless check of event flags and assert in UART_RTOS_Receive.
  - Always waited for RX event flag in UART_RTOS_Receive.

[2.1.0]

- Improvements
  - Added transactional API.

[2.0.0]

- Initial version.

UART_EDMA

[2.5.2]

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

[2.5.1]

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

[2.5.0]

- Refer UART driver change log 2.1.0 to 2.5.0

VREF

[2.1.3]

- Improvements
  - Add timeout for APIs with dfmea issues.

[2.1.2]

- Bug Fixes
  - Fixed the violation of MISRA-2012 rule 10.3.
  - Fixed MISRA C-2012 rule 10.3, rule 10.4 violation.

[2.1.1]

- Bug Fixes
  - MISRA-2012 issue fixed.
    - Fixed rules containing: rule-10.4, rule-10.3, rule-10.1.

[2.1.0]

- Improvements
  - Added new functions to support L5K board: added VREF_SetTrim2V1Val() and VREF_GetTrim2V1Val() functions to supply 2V1 output mode.

[2.0.0]

- Initial version.

WDOG

[2.0.3]

- Bug Fixes
  - Fixed CERT INT31-C violations in WDOG_Init, WDOG_Disable, WDOG_EnableInterrupts and WDOG_DisableInterrupts (MCUX-87729).

[2.0.2]

- Improvements
  - WDG_Init() adds a 256 bus clock delay for WCT window finish.

[2.0.1]

- Bug Fixes
  - MISRA C-2012 issue fixed: rule 10.3, 10.4, 10.6, 10.7 11.9 and 17.7.

[2.0.0]

- Initial version.

1.7 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.

MK22F51212

1.8 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.

1.8.1 Multicore

Multicore SDK

1.8.2 FreeMASTER

freemaster

1.8.3 FreeRTOS

FreeRTOS

1.8.4 File systemFatfs

FatFs

Chapter 2

MK22F51212

2.1 ADC16: 16-bit SAR Analog-to-Digital Converter Driver

void ADC16_Init(ADC_Type *base, const *adc16_config_t* *config)

Initializes the ADC16 module.

Parameters

- base – ADC16 peripheral base address.
- config – Pointer to configuration structure. See “adc16_config_t”.

void ADC16_Deinit(ADC_Type *base)

De-initializes the ADC16 module.

Parameters

- base – ADC16 peripheral base address.

void ADC16_GetDefaultConfig(*adc16_config_t* *config)

Gets an available pre-defined settings for the converter’s configuration.

This function initializes the converter configuration structure with available settings. The default values are as follows.

```
config->referenceVoltageSource      = kADC16_ReferenceVoltageSourceVref;
config->clockSource                 = kADC16_ClockSourceAsynchronousClock;
config->enableAsynchronousClock     = false;
config->clockDivider                = kADC16_ClockDivider8;
config->resolution                  = kADC16_ResolutionSE12Bit;
config->longSampleMode              = kADC16_LongSampleDisabled;
config->enableHighSpeed             = false;
config->enableLowPower              = false;
config->enableContinuousConversion  = false;
```

Parameters

- config – Pointer to the configuration structure.

status_t ADC16_DoAutoCalibration(ADC_Type *base)

Automates the hardware calibration.

This auto calibration helps to adjust the plus/minus side gain automatically. Execute the calibration before using the converter. Note that the hardware trigger should be used during the calibration.

Parameters

- base – ADC16 peripheral base address.

Return values

- kStatus_Success – Calibration is done successfully.
- kStatus_Fail – Calibration has failed.

Returns
Execution status.

static inline void ADC16_SetOffsetValue(ADC_Type *base, int16_t value)

Sets the offset value for the conversion result.

This offset value takes effect on the conversion result. If the offset value is not zero, the reading result is subtracted by it. Note, the hardware calibration fills the offset value automatically.

Parameters

- base – ADC16 peripheral base address.
- value – Setting offset value.

static inline void ADC16_EnableDMA(ADC_Type *base, bool enable)

Enables generating the DMA trigger when the conversion is complete.

Parameters

- base – ADC16 peripheral base address.
- enable – Switcher of the DMA feature. “true” means enabled, “false” means not enabled.

static inline void ADC16_EnableHardwareTrigger(ADC_Type *base, bool enable)

Enables the hardware trigger mode.

Parameters

- base – ADC16 peripheral base address.
- enable – Switcher of the hardware trigger feature. “true” means enabled, “false” means not enabled.

void ADC16_SetChannelMuxMode(ADC_Type *base, *adc16_channel_mux_mode_t* mode)

Sets the channel mux mode.

Some sample pins share the same channel index. The channel mux mode decides which pin is used for an indicated channel.

Parameters

- base – ADC16 peripheral base address.
- mode – Setting channel mux mode. See “adc16_channel_mux_mode_t”.

void ADC16_SetHardwareCompareConfig(ADC_Type *base, const *adc16_hardware_compare_config_t* *config)

Configures the hardware compare mode.

The hardware compare mode provides a way to process the conversion result automatically by using hardware. Only the result in the compare range is available. To compare the range, see “adc16_hardware_compare_mode_t” or the appopriate reference manual for more information.

Parameters

- base – ADC16 peripheral base address.

- config – Pointer to the "adc16_hardware_compare_config_t" structure. Passing "NULL" disables the feature.

void ADC16_SetHardwareAverage(ADC_Type *base, *adc16_hardware_average_mode_t* mode)

Sets the hardware average mode.

The hardware average mode provides a way to process the conversion result automatically by using hardware. The multiple conversion results are accumulated and averaged internally making them easier to read.

Parameters

- base – ADC16 peripheral base address.
- mode – Setting the hardware average mode. See "adc16_hardware_average_mode_t".

void ADC16_SetPGAConfig(ADC_Type *base, const *adc16_pga_config_t* *config)

Configures the PGA for the converter's front end.

Parameters

- base – ADC16 peripheral base address.
- config – Pointer to the "adc16_pga_config_t" structure. Passing "NULL" disables the feature.

uint32_t ADC16_GetStatusFlags(ADC_Type *base)

Gets the status flags of the converter.

Parameters

- base – ADC16 peripheral base address.

Returns

Flags' mask if indicated flags are asserted. See "_adc16_status_flags".

void ADC16_ClearStatusFlags(ADC_Type *base, uint32_t mask)

Clears the status flags of the converter.

Parameters

- base – ADC16 peripheral base address.
- mask – Mask value for the cleared flags. See "_adc16_status_flags".

static inline void ADC16_EnableAsynchronousClockOutput(ADC_Type *base, bool enable)

Enable/disable ADC Asynchronous clock output to other modules.

Parameters

- base – ADC16 peripheral base address.
- enable – Used to enable/disable ADC ADACK output.
  - **true** Asynchronous clock and clock output is enabled regardless of the state of the ADC.
  - **false** Asynchronous clock output disabled, asynchronous clock is enabled only if it is selected as input clock and a conversion is active.

void ADC16_SetChannelConfig(ADC_Type *base, uint32_t channelGroup, const *adc16_channel_config_t* *config)

Configures the conversion channel.

This operation triggers the conversion when in software trigger mode. When in hardware trigger mode, this API configures the channel while the external trigger source helps to trigger the conversion.

Note that the "Channel Group" has a detailed description. To allow sequential conversions of the ADC to be triggered by internal peripherals, the ADC has more than one group of status and control registers, one for each conversion. The channel group parameter indicates which group of registers are used, for example, channel group 0 is for Group A registers and channel group 1 is for Group B registers. The channel groups are used in a "ping-pong" approach to control the ADC operation. At any point, only one of the channel groups is actively controlling ADC conversions. The channel group 0 is used for both software and hardware trigger modes. Channel group 1 and greater indicates multiple channel group registers for use only in hardware trigger mode. See the chip configuration information in the appropriate MCU reference manual for the number of SC1n registers (channel groups) specific to this device. Channel group 1 or greater are not used for software trigger operation. Therefore, writing to these channel groups does not initiate a new conversion. Updating the channel group 0 while a different channel group is actively controlling a conversion is allowed and vice versa. Writing any of the channel group registers while that specific channel group is actively controlling a conversion aborts the current conversion.

Parameters

- base – ADC16 peripheral base address.
- channelGroup – Channel group index.
- config – Pointer to the "adc16_channel_config_t" structure for the conversion channel.

static inline uint32_t ADC16_GetChannelConversionValue(ADC_Type *base, uint32_t channelGroup)

Gets the conversion value.

Parameters

- base – ADC16 peripheral base address.
- channelGroup – Channel group index.

Returns

Conversion value.

uint32_t ADC16_GetChannelStatusFlags(ADC_Type *base, uint32_t channelGroup)

Gets the status flags of channel.

Parameters

- base – ADC16 peripheral base address.
- channelGroup – Channel group index.

Returns

Flags' mask if indicated flags are asserted. See "_adc16_channel_status_flags".

FSL_ADC16_DRIVER_VERSION

ADC16 driver version 2.3.0.

enum _adc16_channel_status_flags

Channel status flags.

Values:

enumerator kADC16_ChannelConversionDoneFlag

Conversion done.

enum _adc16_status_flags

Converter status flags.

Values:

enumerator kADC16_ActiveFlag
Converter is active.

enumerator kADC16_CalibrationFailedFlag
Calibration is failed.

enum _adc_channel_mux_mode
Channel multiplexer mode for each channel.

For some ADC16 channels, there are two pin selections in channel multiplexer. For example, ADC0_SE4a and ADC0_SE4b are the different channels that share the same channel number.

Values:

enumerator kADC16_ChannelMuxA
For channel with channel mux a.

enumerator kADC16_ChannelMuxB
For channel with channel mux b.

enum _adc16_clock_divider
Clock divider for the converter.

Values:

enumerator kADC16_ClockDivider1
For divider 1 from the input clock to the module.

enumerator kADC16_ClockDivider2
For divider 2 from the input clock to the module.

enumerator kADC16_ClockDivider4
For divider 4 from the input clock to the module.

enumerator kADC16_ClockDivider8
For divider 8 from the input clock to the module.

enum _adc16_resolution
Converter's resolution.

Values:

enumerator kADC16_Resolution8or9Bit
Single End 8-bit or Differential Sample 9-bit.

enumerator kADC16_Resolution12or13Bit
Single End 12-bit or Differential Sample 13-bit.

enumerator kADC16_Resolution10or11Bit
Single End 10-bit or Differential Sample 11-bit.

enumerator kADC16_ResolutionSE8Bit
Single End 8-bit.

enumerator kADC16_ResolutionSE12Bit
Single End 12-bit.

enumerator kADC16_ResolutionSE10Bit
Single End 10-bit.

enumerator kADC16_ResolutionDF9Bit
Differential Sample 9-bit.

enumerator kADC16_ResolutionDF13Bit
Differential Sample 13-bit.

enumerator kADC16_ResolutionDF11Bit
Differential Sample 11-bit.

enum _adc16_clock_source
Clock source.
Values:

enumerator kADC16_ClockSourceAlt0
Selection 0 of the clock source.

enumerator kADC16_ClockSourceAlt1
Selection 1 of the clock source.

enumerator kADC16_ClockSourceAlt2
Selection 2 of the clock source.

enumerator kADC16_ClockSourceAlt3
Selection 3 of the clock source.

enumerator kADC16_ClockSourceAsynchronousClock
Using internal asynchronous clock.

enum _adc16_long_sample_mode
Long sample mode.
Values:

enumerator kADC16_LongSampleCycle24
20 extra ADCK cycles, 24 ADCK cycles total.

enumerator kADC16_LongSampleCycle16
12 extra ADCK cycles, 16 ADCK cycles total.

enumerator kADC16_LongSampleCycle10
6 extra ADCK cycles, 10 ADCK cycles total.

enumerator kADC16_LongSampleCycle6
2 extra ADCK cycles, 6 ADCK cycles total.

enumerator kADC16_LongSampleDisabled
Disable the long sample feature.

enum _adc16_reference_voltage_source
Reference voltage source.
Values:

enumerator kADC16_ReferenceVoltageSourceVref
For external pins pair of VrefH and VrefL.

enumerator kADC16_ReferenceVoltageSourceValt
For alternate reference pair of ValtH and ValtL.

enum _adc16_hardware_average_mode
Hardware average mode.
Values:

enumerator kADC16_HardwareAverageCount4
For hardware average with 4 samples.

enumerator kADC16_HardwareAverageCount8
For hardware average with 8 samples.

enumerator kADC16_HardwareAverageCount16
For hardware average with 16 samples.

enumerator kADC16_HardwareAverageCount32
For hardware average with 32 samples.

enumerator kADC16_HardwareAverageDisabled
Disable the hardware average feature.

enum _adc16_hardware_compare_mode
Hardware compare mode.
Values:

enumerator kADC16_HardwareCompareMode0
x < value1.

enumerator kADC16_HardwareCompareMode1
x > value1.

enumerator kADC16_HardwareCompareMode2
if value1 <= value2, then x < value1 || x > value2; else, value1 > x > value2.

enumerator kADC16_HardwareCompareMode3
if value1 <= value2, then value1 <= x <= value2; else x >= value1 || x <= value2.

enum _adc16_pga_gain
PGA's Gain mode.
Values:

enumerator kADC16_PGAGainValueOf1
For amplifier gain of 1.

enumerator kADC16_PGAGainValueOf2
For amplifier gain of 2.

enumerator kADC16_PGAGainValueOf4
For amplifier gain of 4.

enumerator kADC16_PGAGainValueOf8
For amplifier gain of 8.

enumerator kADC16_PGAGainValueOf16
For amplifier gain of 16.

enumerator kADC16_PGAGainValueOf32
For amplifier gain of 32.

enumerator kADC16_PGAGainValueOf64
For amplifier gain of 64.

typedef enum *_adc_channel_mux_mode* adc16_channel_mux_mode_t
Channel multiplexer mode for each channel.
For some ADC16 channels, there are two pin selections in channel multiplexer. For example, ADC0_SE4a and ADC0_SE4b are the different channels that share the same channel number.

typedef enum *_adc16_clock_divider* adc16_clock_divider_t
Clock divider for the converter.

typedef enum *_adc16_resolution* adc16_resolution_t
Converter's resolution.

typedef enum *_adc16_clock_source* adc16_clock_source_t
Clock source.

typedef enum *_adc16_long_sample_mode* adc16_long_sample_mode_t
Long sample mode.

typedef enum *_adc16_reference_voltage_source* adc16_reference_voltage_source_t
Reference voltage source.

typedef enum *_adc16_hardware_average_mode* adc16_hardware_average_mode_t
Hardware average mode.

typedef enum *_adc16_hardware_compare_mode* adc16_hardware_compare_mode_t
Hardware compare mode.

typedef enum *_adc16_pga_gain* adc16_pga_gain_t
PGA's Gain mode.

typedef struct *_adc16_config* adc16_config_t
ADC16 converter configuration.

typedef struct *_adc16_hardware_compare_config* adc16_hardware_compare_config_t
ADC16 Hardware comparison configuration.

typedef struct *_adc16_channel_config* adc16_channel_config_t
ADC16 channel conversion configuration.

typedef struct *_adc16_pga_config* adc16_pga_config_t
ADC16 programmable gain amplifier configuration.

struct _adc16_config
#include <fsl_adc16.h> ADC16 converter configuration.

Public Members

adc16_reference_voltage_source_t referenceVoltageSource
Select the reference voltage source.

adc16_clock_source_t clockSource
Select the input clock source to converter.

bool enableAsynchronousClock
Enable the asynchronous clock output.

adc16_clock_divider_t clockDivider
Select the divider of input clock source.

adc16_resolution_t resolution
Select the sample resolution mode.

adc16_long_sample_mode_t longSampleMode
Select the long sample mode.

bool enableHighSpeed
Enable the high-speed mode.

bool enableLowPower
Enable low power.

bool enableContinuousConversion
Enable continuous conversion mode.

adc16_hardware_average_mode_t hardwareAverageMode
Set hardware average mode.

struct _adc16_hardware_compare_config
#include <fsl_adc16.h> ADC16 Hardware comparison configuration.

Public Members

adc16_hardware_compare_mode_t hardwareCompareMode
Select the hardware compare mode. See “adc16_hardware_compare_mode_t”.

int16_t value1
Setting value1 for hardware compare mode.

int16_t value2
Setting value2 for hardware compare mode.

struct _adc16_channel_config
#include <fsl_adc16.h> ADC16 channel conversion configuration.

Public Members

uint32_t channelNumber
Setting the conversion channel number. The available range is 0-31. See channel connection information for each chip in Reference Manual document.

bool enableInterruptOnConversionCompleted
Generate an interrupt request once the conversion is completed.

bool enableDifferentialConversion
Using Differential sample mode.

struct _adc16_pga_config
#include <fsl_adc16.h> ADC16 programmable gain amplifier configuration.

Public Members

adc16_pga_gain_t pgaGain
Setting PGA gain.

bool enableRunInNormalMode
Enable PGA working in normal mode, or low power mode by default.

bool disablePgaChopping
Disable the PGA chopping function. The PGA employs chopping to remove/reduce offset and 1/f noise and offers an offset measurement configuration that aids the offset calibration.

bool enableRunInOffsetMeasurement
Enable the PGA working in offset measurement mode. When this feature is enabled, the PGA disconnects itself from the external inputs and auto-configures into offset measurement mode. With this field set, run the ADC in the recommended settings and enable the maximum hardware averaging to get the PGA offset number. The output is the (PGA offset * (64+1)) for the given PGA setting.

2.2 Clock Driver

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

enumerator kCLOCK_CoreSysClk
Core/system clock

enumerator kCLOCK_PlatClk
Platform clock

enumerator kCLOCK_BusClk
Bus clock

enumerator kCLOCK_FlexBusClk
FlexBus clock

enumerator kCLOCK_FlashClk
Flash clock

enumerator kCLOCK_FastPeriphClk
Fast peripheral clock

enumerator kCLOCK_PllFllSelClk
The clock after SIM[PLLFLLSEL].

enumerator kCLOCK_Er32kClk
External reference 32K clock (ERCLK32K)

enumerator kCLOCK_Osc0ErClk
OSC0 external reference clock (OSC0ERCLK)

enumerator kCLOCK_Osc1ErClk
OSC1 external reference clock (OSC1ERCLK)

enumerator kCLOCK_Osc0ErClkUndiv
OSC0 external reference undivided clock(OSC0ERCLK_UNDIV).

enumerator kCLOCK_McgFixedFreqClk
MCG fixed frequency clock (MCGFFCLK)

enumerator kCLOCK_McgInternalRefClk
MCG internal reference clock (MCGIRCLK)

enumerator kCLOCK_McgFllClk
MCGFLLCLK

enumerator kCLOCK_McgPll0Clk
MCGPLL0CLK

enumerator kCLOCK_McgPll1Clk
MCGPLL1CLK

enumerator kCLOCK_McgExtPllClk
EXT_PLLCLK

enumerator kCLOCK_McgPeriphClk
MCG peripheral clock (MCGPCLK)

enumerator kCLOCK_McgIrc48MClk
MCG IRC48M clock

enumerator kCLOCK_LpoClk
LPO clock

enum _clock_usb_src
USB clock source definition.
Values:

enumerator kCLOCK_UsbSrcPll0
Use PLL0.

enumerator kCLOCK_UsbSrcIrc48M
Use IRC48M.

enumerator kCLOCK_UsbSrcExt
Use USB_CLKIN.

enum _clock_ip_name
Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.
Values:

enumerator kCLOCK_IpInvalid

enumerator kCLOCK_Ewm0

enumerator kCLOCK_I2c0

enumerator kCLOCK_I2c1

enumerator kCLOCK_Uart0

enumerator kCLOCK_Uart1

enumerator kCLOCK_Uart2

enumerator kCLOCK_Usbfs0

enumerator kCLOCK_Cmp0

enumerator kCLOCK_Cmp1

enumerator kCLOCK_Vref0

enumerator kCLOCK_Lptmr0

enumerator kCLOCK_PortA

enumerator kCLOCK_PortB

enumerator kCLOCK_PortC

enumerator kCLOCK_PortD

enumerator kCLOCK_PortE

enumerator kCLOCK_Ftf0

enumerator kCLOCK_Dmamux0

enumerator kCLOCK_Ftm3

enumerator kCLOCK_Adc1

enumerator kCLOCK_Dac1

enumerator kCLOCK_Rnga0

enumerator kCLOCK_Lpuart0

enumerator kCLOCK_Spi0

enumerator kCLOCK_Spi1

enumerator kCLOCK_Sai0

enumerator kCLOCK_Crc0

enumerator kCLOCK_Pdb0

enumerator kCLOCK_Pit0

enumerator kCLOCK_Ftm0

enumerator kCLOCK_Ftm1

enumerator kCLOCK_Ftm2

enumerator kCLOCK_Adc0

enumerator kCLOCK_Rtc0

enumerator kCLOCK_Dac0

enumerator kCLOCK_Flexbus0

enumerator kCLOCK_Dma0

enum _osc_mode

OSC work mode.

Values:

enumerator kOSC_ModeExt

Use an external clock.

enumerator kOSC_ModeOscLowPower
Oscillator low power.

enumerator kOSC_ModeOscHighGain
Oscillator high gain.

enum _osc_cap_load
Oscillator capacitor load setting.
Values:

enumerator kOSC_Cap2P
2 pF capacitor load

enumerator kOSC_Cap4P
4 pF capacitor load

enumerator kOSC_Cap8P
8 pF capacitor load

enumerator kOSC_Cap16P
16 pF capacitor load

enum _oscer_enable_mode
OSCERCLK enable mode.
Values:

enumerator kOSC_ErClkEnable
Enable.

enumerator kOSC_ErClkEnableInStop
Enable in stop mode.

enum _mcg_fll_src
MCG FLL reference clock source select.
Values:

enumerator kMCG_FllSrcExternal
External reference clock is selected

enumerator kMCG_FllSrcInternal
The slow internal reference clock is selected

enum _mcg_irc_mode
MCG internal reference clock select.
Values:

enumerator kMCG_IrcSlow
Slow internal reference clock selected

enumerator kMCG_IrcFast
Fast internal reference clock selected

enum _mcg_dmx32
MCG DCO Maximum Frequency with 32.768 kHz Reference.
Values:

enumerator kMCG_Dmx32Default
DCO has a default range of 25%

enumerator kMCG_Dmx32Fine
DCO is fine-tuned for maximum frequency with 32.768 kHz reference

enum _mcg_drs
MCG DCO range select.
Values:

enumerator kMCG_DrsLow
Low frequency range

enumerator kMCG_DrsMid
Mid frequency range

enumerator kMCG_DrsMidHigh
Mid-High frequency range

enumerator kMCG_DrsHigh
High frequency range

enum _mcg_pll_ref_src
MCG PLL reference clock select.
Values:

enumerator kMCG_PllRefOsc0
Selects OSC0 as PLL reference clock

enumerator kMCG_PllRefOsc1
Selects OSC1 as PLL reference clock

enum _mcg_clkout_src
MCGOUT clock source.
Values:

enumerator kMCG_ClkOutSrcOut
Output of the FLL is selected (reset default)

enumerator kMCG_ClkOutSrcInternal
Internal reference clock is selected

enumerator kMCG_ClkOutSrcExternal
External reference clock is selected

enum _mcg_atm_select
MCG Automatic Trim Machine Select.
Values:

enumerator kMCG_AtmSel32k
32 kHz Internal Reference Clock selected

enumerator kMCG_AtmSel4m
4 MHz Internal Reference Clock selected

enum _mcg_oscsel
MCG OSC Clock Select.
Values:

enumerator kMCG_OscselOsc
Selects System Oscillator (OSCCLK)

enumerator kMCG_OscselRtc
Selects 32 kHz RTC Oscillator

enumerator kMCG_OscselIrc
Selects 48 MHz IRC Oscillator

enum _mcg_pll_clk_select
MCG PLLCS select.
Values:

enumerator kMCG_PllClkSelPll0
PLL0 output clock is selected

enumerator kMCG_PllClkSelPll1

enum _mcg_monitor_mode
MCG clock monitor mode.
Values:

enumerator kMCG_MonitorNone
Clock monitor is disabled.

enumerator kMCG_MonitorInt
Trigger interrupt when clock lost.

enumerator kMCG_MonitorReset
System reset when clock lost.

MCG status. Enumeration _mcg_status.

Values:

enumerator kStatus_MCG_ModeUnreachable
Can't switch to target mode.

enumerator kStatus_MCG_ModeInvalid
Current mode invalid for the specific function.

enumerator kStatus_MCG_AtmBusClockInvalid
Invalid bus clock for ATM.

enumerator kStatus_MCG_AtmDesiredFreqInvalid
Invalid desired frequency for ATM.

enumerator kStatus_MCG_AtmIrcUsed
IRC is used when using ATM.

enumerator kStatus_MCG_AtmHardwareFail
Hardware fail occurs during ATM.

enumerator kStatus_MCG_SourceUsed
Can't change the clock source because it is in use.

MCG status flags. Enumeration _mcg_status_flags_t.

Values:

enumerator kMCG_Osc0LostFlag
OSC0 lost.

enumerator kMCG_Osc0InitFlag
OSC0 crystal initialized.

enumerator kMCG_RtcOscLostFlag
RTC OSC lost.

enumerator kMCG_Pll0LostFlag
PLL0 lost.

enumerator kMCG_Pll0LockFlag
PLL0 locked.

MCG internal reference clock (MCGIRCLK) enable mode definition. Enumeration _mcg_irclk_enable_mode.

Values:

enumerator kMCG_IrclkEnable
MCGIRCLK enable.

enumerator kMCG_IrclkEnableInStop
MCGIRCLK enable in stop mode.

MCG PLL clock enable mode definition. Enumeration _mcg_pll_enable_mode.

Values:

enumerator kMCG_PllEnableIndependent
MCGPLLCLK enable independent of the MCG clock mode. Generally, the PLL is disabled in FLL modes (FEI/FBI/FEE/FBE). Setting the PLL clock enable independent, enables the PLL in the FLL modes.

enumerator kMCG_PllEnableInStop
MCGPLLCLK enable in STOP mode.

enum _mcg_mode
MCG mode definitions.

Values:

enumerator kMCG_ModeFEI
FEI - FLL Engaged Internal

enumerator kMCG_ModeFBI
FBI - FLL Bypassed Internal

enumerator kMCG_ModeBLPI
BLPI - Bypassed Low Power Internal

enumerator kMCG_ModeFEE
FEE - FLL Engaged External

enumerator kMCG_ModeFBE
FBE - FLL Bypassed External

enumerator kMCG_ModeBLPE
BLPE - Bypassed Low Power External

enumerator kMCG_ModePBE
PBE - PLL Bypassed External

enumerator kMCG_ModePEE
PEE - PLL Engaged External

enumerator kMCG_ModeError
Unknown mode

typedef enum *_clock_name* clock_name_t

Clock name used to get clock frequency.

typedef enum *_clock_usb_src* clock_usb_src_t

USB clock source definition.

typedef enum *_clock_ip_name* clock_ip_name_t

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

typedef struct *_sim_clock_config* sim_clock_config_t

SIM configuration structure for clock setting.

typedef enum *_osc_mode* osc_mode_t

OSC work mode.

typedef struct *_oscer_config* oscer_config_t

OSC configuration for OSCERCLK.

typedef struct *_osc_config* osc_config_t

OSC Initialization Configuration Structure.

Defines the configuration data structure to initialize the OSC. When porting to a new board, set the following members according to the board setting:

a. freq: The external frequency.

b. workMode: The OSC module mode.

typedef enum *_mcg_fll_src* mcg_fll_src_t

MCG FLL reference clock source select.

typedef enum *_mcg_irc_mode* mcg_irc_mode_t

MCG internal reference clock select.

typedef enum *_mcg_dmx32* mcg_dmx32_t

MCG DCO Maximum Frequency with 32.768 kHz Reference.

typedef enum *_mcg_drs* mcg_drs_t

MCG DCO range select.

typedef enum *_mcg_pll_ref_src* mcg_pll_ref_src_t

MCG PLL reference clock select.

typedef enum *_mcg_clkout_src* mcg_clkout_src_t

MCGOUT clock source.

typedef enum *_mcg_atm_select* mcg_atm_select_t

MCG Automatic Trim Machine Select.

typedef enum *_mcg_oscsel* mcg_oscsel_t

MCG OSC Clock Select.

typedef enum *_mcg_pll_clk_select* mcg_pll_clk_select_t

MCG PLLCS select.

typedef enum *_mcg_monitor_mode* mcg_monitor_mode_t

MCG clock monitor mode.

typedef enum *_mcg_mode* mcg_mode_t

MCG mode definitions.

typedef struct *_mcg_pll_config* mcg_pll_config_t

MCG PLL configuration.

typedef struct *_mcg_config* mcg_config_t

MCG mode change configuration structure.

When porting to a new board, set the following members according to the board setting:

a. frdiv: If the FLL uses the external reference clock, set this value to ensure that the external reference clock divided by frdiv is in the 31.25 kHz to 39.0625 kHz range.

b. The PLL reference clock divider PRDIV: PLL reference clock frequency after PRDIV should be in the FSL_FEATURE_MCG_PLL_REF_MIN to FSL_FEATURE_MCG_PLL_REF_MAX range.

volatile uint32_t g_xtal0Freq

External XTAL0 (OSC0) clock frequency.

The XTAL0/EXTAL0 (OSC0) clock frequency in Hz. When the clock is set up, use the function CLOCK_SetXtal0Freq to set the value in the clock driver. For example, if XTAL0 is 8 MHz:

```
Set up the OSC0
CLOCK_InitOsc0(...);
Set the XTAL0 value to the clock driver.
CLOCK_SetXtal0Freq(80000000);
```

This is important for the multicore platforms where only one core needs to set up the OSC0 using the CLOCK_InitOsc0. All other cores need to call the CLOCK_SetXtal0Freq to get a valid clock frequency.

volatile uint32_t g_xtal32Freq

External XTAL32/EXTAL32/RTC_CLKIN clock frequency.

The XTAL32/EXTAL32/RTC_CLKIN clock frequency in Hz. When the clock is set up, use the function CLOCK_SetXtal32Freq to set the value in the clock driver.

This is important for the multicore platforms where only one core needs to set up the clock. All other cores need to call the CLOCK_SetXtal32Freq to get a valid clock frequency.

static inline void CLOCK_EnableClock(*clock_ip_name_t* name)

Enable the clock for specific IP.

Parameters

- name – Which clock to enable, see *clock_ip_name_t*.

static inline void CLOCK_DisableClock(*clock_ip_name_t* name)

Disable the clock for specific IP.

Parameters

- name – Which clock to disable, see *clock_ip_name_t*.

static inline void CLOCK_SetLpuartClock(uint32_t src)

Set LPUART clock source.

Parameters

- src – The value to set LPUART clock source.

static inline void CLOCK_SetEr32kClock(uint32_t src)

Set ERCLK32K source.

Parameters

- src – The value to set ERCLK32K clock source.

static inline void CLOCK_SetTraceClock(uint32_t src)

Set debug trace clock source.

Parameters

- src – The value to set debug trace clock source.

static inline void CLOCK_SetPllFllSelClock(uint32_t src)

Set PLLFLLSEL clock source.

Parameters

- src – The value to set PLLFLLSEL clock source.

static inline void CLOCK_SetClkOutClock(uint32_t src)

Set CLKOUT source.

Parameters

- src – The value to set CLKOUT source.

static inline void CLOCK_SetRtcClkOutClock(uint32_t src)

Set RTC_CLKOUT source.

Parameters

- src – The value to set RTC_CLKOUT source.

bool CLOCK_EnableUsbfs0Clock(*clock_usb_src_t* src, uint32_t freq)

Enable USB FS clock.

Parameters

- src – USB FS clock source.
- freq – The frequency specified by src.

Return values

- true – The clock is set successfully.
- false – The clock source is invalid to get proper USB FS clock.

static inline void CLOCK_DisableUsbfs0Clock(void)

Disable USB FS clock.

Disable USB FS clock.

static inline void CLOCK_SetOutDiv(uint32_t outdiv1, uint32_t outdiv2, uint32_t outdiv3, uint32_t outdiv4)

System clock divider.

Set the SIM_CLKDIV1[OUTDIV1], SIM_CLKDIV1[OUTDIV2], SIM_CLKDIV1[OUTDIV3], SIM_CLKDIV1[OUTDIV4].

Parameters

- outdiv1 – Clock 1 output divider value.
- outdiv2 – Clock 2 output divider value.
- outdiv3 – Clock 3 output divider value.
- outdiv4 – Clock 4 output divider value.

uint32_t CLOCK_GetFreq(*clock_name_t* clockName)

Gets the clock frequency for a specific clock name.

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in clock_name_t. The MCG must be properly configured before using this function.

Parameters

- clockName – Clock names defined in clock_name_t

Returns

Clock frequency value in Hertz

uint32_t CLOCK_GetCoreSysClkFreq(void)

Get the core clock or system clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetPlatClkFreq(void)

Get the platform clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetBusClkFreq(void)

Get the bus clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetFlexBusClkFreq(void)

Get the flexbus clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetFlashClkFreq(void)

Get the flash clock frequency.

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetPllFllSelClkFreq(void)

Get the output clock frequency selected by SIM[PLLFLLSEL].

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetEr32kClkFreq(void)

Get the external reference 32K clock frequency (ERCLK32K).

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetOsc0ErClkUndivFreq(void)

Get the OSC0 external reference undivided clock frequency (OSC0ERCLK_UNDIV).

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetOsc0ErClkFreq(void)

Get the OSC0 external reference clock frequency (OSC0ERCLK).

Returns

Clock frequency in Hz.

uint32_t CLOCK_GetOsc0ErClkDivFreq(void)

Get the OSC0 external reference divided clock frequency.

Returns

Clock frequency in Hz.

void CLOCK_SetSimConfig(*sim_clock_config_t* const *config)

Set the clock configure in SIM module.

This function sets system layer clock settings in SIM module.

Parameters

- config – Pointer to the configure structure.

static inline void CLOCK_SetSimSafeDivs(void)

Set the system clock dividers in SIM to safe value.

The system level clocks (core clock, bus clock, flexbus clock and flash clock) must be in allowed ranges. During MCG clock mode switch, the MCG output clock changes then the system level clocks may be out of range. This function could be used before MCG mode change, to make sure system level clocks are in allowed range.

FSL_CLOCK_DRIVER_VERSION

CLOCK driver version 2.5.3.

MCG_INTERNAL_IRC_48M

IRC48M clock frequency in Hz.

SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY

DMAMUX_CLOCKS

Clock ip name array for DMAMUX.

RTC_CLOCKS

Clock ip name array for RTC.

SAI_CLOCKS

Clock ip name array for SAI.

PORT_CLOCKS

Clock ip name array for PORT.

FLEXBUS_CLOCKS

Clock ip name array for FLEXBUS.

EWM_CLOCKS
: Clock ip name array for EWM.

PIT_CLOCKS
: Clock ip name array for PIT.

DSPI_CLOCKS
: Clock ip name array for DSPI.

LPTMR_CLOCKS
: Clock ip name array for LPTMR.

FTM_CLOCKS
: Clock ip name array for FTM.

EDMA_CLOCKS
: Clock ip name array for EDMA.

LPUART_CLOCKS
: Clock ip name array for LPUART.

DAC_CLOCKS
: Clock ip name array for DAC.

ADC16_CLOCKS
: Clock ip name array for ADC16.

VREF_CLOCKS
: Clock ip name array for VREF.

UART_CLOCKS
: Clock ip name array for UART.

RNGA_CLOCKS
: Clock ip name array for RNGA.

CRC_CLOCKS
: Clock ip name array for CRC.

I2C_CLOCKS
: Clock ip name array for I2C.

FTF_CLOCKS
: Clock ip name array for FTF.

PDB_CLOCKS
: Clock ip name array for PDB.

CMP_CLOCKS
: Clock ip name array for CMP.

LPO_CLK_FREQ
: LPO clock frequency.

SYS_CLK
: Peripherals clock source definition.

BUS_CLK

FAST_CLK

I2C0_CLK_SRC

I2C1_CLK_SRC

DSPI0_CLK_SRC

DSPI1_CLK_SRC

UART0_CLK_SRC

UART1_CLK_SRC

UART2_CLK_SRC

CLK_GATE_REG_OFFSET_SHIFT

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)

uint32_t CLOCK_GetOutClkFreq(void)
: Gets the MCG output clock (MCGOUTCLK) frequency.

  This function gets the MCG output clock frequency in Hz based on the current MCG register value.

  Returns
  The frequency of MCGOUTCLK.

uint32_t CLOCK_GetFllFreq(void)

Gets the MCG FLL clock (MCGFLLCLK) frequency.

This function gets the MCG FLL clock frequency in Hz based on the current MCG register value. The FLL is enabled in FEI/FBI/FEE/FBE mode and disabled in low power state in other modes.

Returns

The frequency of MCGFLLCLK.

uint32_t CLOCK_GetInternalRefClkFreq(void)

Gets the MCG internal reference clock (MCGIRCLK) frequency.

This function gets the MCG internal reference clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGIRCLK.

uint32_t CLOCK_GetFixedFreqClkFreq(void)

Gets the MCG fixed frequency clock (MCGFFCLK) frequency.

This function gets the MCG fixed frequency clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGFFCLK.

uint32_t CLOCK_GetPll0Freq(void)

Gets the MCG PLL0 clock (MCGPLL0CLK) frequency.

This function gets the MCG PLL0 clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGPLL0CLK.

static inline void CLOCK_SetLowPowerEnable(bool enable)

Enables or disables the MCG low power.

Enabling the MCG low power disables the PLL and FLL in bypass modes. In other words, in FBE and PBE modes, enabling low power sets the MCG to BLPE mode. In FBI and PBI modes, enabling low power sets the MCG to BLPI mode. When disabling the MCG low power, the PLL or FLL are enabled based on MCG settings.

Parameters

- enable – True to enable MCG low power, false to disable MCG low power.

status_t CLOCK_SetInternalRefClkConfig(uint8_t enableMode, *mcg_irc_mode_t* ircs, uint8_t fcrdiv)

Configures the Internal Reference clock (MCGIRCLK).

This function sets the MCGIRCLK base on parameters. It also selects the IRC source. If the fast IRC is used, this function sets the fast IRC divider. This function also sets whether the MCGIRCLK is enabled in stop mode. Calling this function in FBI/PBI/BLPI modes may change the system clock. As a result, using the function in these modes it is not allowed.

Parameters

- enableMode – MCGIRCLK enable mode, OR'ed value of the enumeration _mcg_irclk_enable_mode.
- ircs – MCGIRCLK clock source, choose fast or slow.
- fcrdiv – Fast IRC divider setting (FCRDIV).

Return values

- kStatus_MCG_SourceUsed – Because the internal reference clock is used as a clock source, the configuration should not be changed. Otherwise, a glitch occurs.
- kStatus_Success – MCGIRCLK configuration finished successfully.

status_t CLOCK_SetExternalRefClkConfig(*mcg_oscsel_t* oscsel)

Selects the MCG external reference clock.

Selects the MCG external reference clock source, changes the MCG_C7[OSCSEL], and waits for the clock source to be stable. Because the external reference clock should not be changed in FEE/FBE/BLPE/PBE/PEE modes, do not call this function in these modes.

Parameters

- oscsel – MCG external reference clock source, MCG_C7[OSCSEL].

Return values

- kStatus_MCG_SourceUsed – Because the external reference clock is used as a clock source, the configuration should not be changed. Otherwise, a glitch occurs.
- kStatus_Success – External reference clock set successfully.

static inline void CLOCK_SetFllExtRefDiv(uint8_t frdiv)

Set the FLL external reference clock divider value.

Sets the FLL external reference clock divider value, the register MCG_C1[FRDIV].

Parameters

- frdiv – The FLL external reference clock divider value, MCG_C1[FRDIV].

void CLOCK_EnablePll0(*mcg_pll_config_t* const *config)

Enables the PLL0 in FLL mode.

This function sets us the PLL0 in FLL mode and reconfigures the PLL0. Ensure that the PLL reference clock is enabled before calling this function and that the PLL0 is not used as a clock source. The function CLOCK_CalcPllDiv gets the correct PLL divider values.

Parameters

- config – Pointer to the configuration structure.

static inline void CLOCK_DisablePll0(void)

Disables the PLL0 in FLL mode.

This function disables the PLL0 in FLL mode. It should be used together with the *CLOCK_EnablePll0*.

uint32_t CLOCK_CalcPllDiv(uint32_t refFreq, uint32_t desireFreq, uint8_t *prdiv, uint8_t *vdiv)

Calculates the PLL divider setting for a desired output frequency.

This function calculates the correct reference clock divider (PRDIV) and VCO divider (VDIV) to generate a desired PLL output frequency. It returns the closest frequency match with the corresponding PRDIV/VDIV returned from parameters. If a desired frequency is not valid, this function returns 0.

Parameters

- refFreq – PLL reference clock frequency.
- desireFreq – Desired PLL output frequency.
- prdiv – PRDIV value to generate desired PLL frequency.
- vdiv – VDIV value to generate desired PLL frequency.

Returns
Closest frequency match that the PLL was able generate.

void CLOCK_SetOsc0MonitorMode(*mcg_monitor_mode_t* mode)

Sets the OSC0 clock monitor mode.

This function sets the OSC0 clock monitor mode. See *mcg_monitor_mode_t* for details.

Parameters

- mode – Monitor mode to set.

void CLOCK_SetRtcOscMonitorMode(*mcg_monitor_mode_t* mode)

Sets the RTC OSC clock monitor mode.

This function sets the RTC OSC clock monitor mode. See *mcg_monitor_mode_t* for details.

Parameters

- mode – Monitor mode to set.

void CLOCK_SetPll0MonitorMode(*mcg_monitor_mode_t* mode)

Sets the PLL0 clock monitor mode.

This function sets the PLL0 clock monitor mode. See *mcg_monitor_mode_t* for details.

Parameters

- mode – Monitor mode to set.

uint32_t CLOCK_GetStatusFlags(void)

Gets the MCG status flags.

This function gets the MCG clock status flags. All status flags are returned as a logical OR of the enumeration refer to _mcg_status_flags_t. To check a specific flag, compare the return value with the flag.

Example:

```
To check the clock lost lock status of OSC0 and PLL0.
uint32_t mcgFlags;

mcgFlags = CLOCK_GetStatusFlags();

if (mcgFlags & kMCG_Osc0LostFlag)
{
    OSC0 clock lock lost. Do something.
}
if (mcgFlags & kMCG_Pll0LostFlag)
{
    PLL0 clock lock lost. Do something.
}
```

Returns
Logical OR value of the enumeration _mcg_status_flags_t.

void CLOCK_ClearStatusFlags(uint32_t mask)

Clears the MCG status flags.

This function clears the MCG clock lock lost status. The parameter is a logical OR value of the flags to clear. See the enumeration _mcg_status_flags_t.

Example:

```
To clear the clock lost lock status flags of OSC0 and PLL0.

CLOCK_ClearStatusFlags(kMCG_Osc0LostFlag | kMCG_Pll0LostFlag);
```

Parameters

- mask – The status flags to clear. This is a logical OR of members of the enumeration _mcg_status_flags_t.

static inline void OSC_SetExtRefClkConfig(OSC_Type *base, *oscer_config_t* const *config)

Configures the OSC external reference clock (OSCERCLK).

This function configures the OSC external reference clock (OSCERCLK). This is an example to enable the OSCERCLK in normal and stop modes and also set the output divider to 1:

```
oscer_config_t config =
{
    .enableMode = kOSC_ErClkEnable | kOSC_ErClkEnableInStop,
    .erclkDiv   = 1U,
};

OSC_SetExtRefClkConfig(OSC, &config);
```

Parameters

- base – OSC peripheral address.
- config – Pointer to the configuration structure.

static inline void OSC_SetCapLoad(OSC_Type *base, uint8_t capLoad)

Sets the capacitor load configuration for the oscillator.

This function sets the specified capacitors configuration for the oscillator. This should be done in the early system level initialization function call based on the system configuration.

Example:

```
To enable only 2 pF and 8 pF capacitor load, please use like this.
OSC_SetCapLoad(OSC, kOSC_Cap2P | kOSC_Cap8P);
```

Parameters

- base – OSC peripheral address.
- capLoad – OR'ed value for the capacitor load option, see *_osc_cap_load*.

void CLOCK_InitOsc0(*osc_config_t* const *config)

Initializes the OSC0.

This function initializes the OSC0 according to the board configuration.

Parameters

- config – Pointer to the OSC0 configuration structure.

void CLOCK_DeinitOsc0(void)

Deinitializes the OSC0.

This function deinitializes the OSC0.

static inline void CLOCK_SetXtal0Freq(uint32_t freq)

Sets the XTAL0 frequency based on board settings.

Parameters

- freq – The XTAL0/EXTAL0 input clock frequency in Hz.

static inline void CLOCK_SetXtal32Freq(uint32_t freq)

Sets the XTAL32/RTC_CLKIN frequency based on board settings.

Parameters

- freq – The XTAL32/EXTAL32/RTC_CLKIN input clock frequency in Hz.

void CLOCK_SetSlowIrcFreq(uint32_t freq)

Set the Slow IRC frequency based on the trimmed value.

Parameters

- freq – The Slow IRC frequency input clock frequency in Hz.

void CLOCK_SetFastIrcFreq(uint32_t freq)

Set the Fast IRC frequency based on the trimmed value.

Parameters

- freq – The Fast IRC frequency input clock frequency in Hz.

status_t CLOCK_TrimInternalRefClk(uint32_t extFreq, uint32_t desireFreq, uint32_t *actualFreq, *mcg_atm_select_t* atms)

Auto trims the internal reference clock.

This function trims the internal reference clock by using the external clock. If successful, it returns the kStatus_Success and the frequency after trimming is received in the parameter actualFreq. If an error occurs, the error code is returned.

Parameters

- extFreq – External clock frequency, which should be a bus clock.
- desireFreq – Frequency to trim to.
- actualFreq – Actual frequency after trimming.
- atms – Trim fast or slow internal reference clock.

Return values

- kStatus_Success – ATM success.
- kStatus_MCG_AtmBusClockInvalid – The bus clock is not in allowed range for the ATM.
- kStatus_MCG_AtmDesiredFreqInvalid – MCGIRCLK could not be trimmed to the desired frequency.
- kStatus_MCG_AtmIrcUsed – Could not trim because MCGIRCLK is used as a bus clock source.
- kStatus_MCG_AtmHardwareFail – Hardware fails while trimming.

mcg_mode_t CLOCK_GetMode(void)

Gets the current MCG mode.

This function checks the MCG registers and determines the current MCG mode.

Returns

Current MCG mode or error code; See *mcg_mode_t*.

status_t CLOCK_SetFeiMode(*mcg_dmx32_t* dmx32, *mcg_drs_t* drs, void (*fllStableDelay)(void))

Sets the MCG to FEI mode.

This function sets the MCG to FEI mode. If setting to FEI mode fails from the current mode, this function returns an error.

Note: If dmx32 is set to kMCG_Dmx32Fine, the slow IRC must not be trimmed to a frequency above 32768 Hz.

Parameters

- dmx32 – DMX32 in FEI mode.
- drs – The DCO range selection.
- fllStableDelay – Delay function to ensure that the FLL is stable. Passing NULL does not cause a delay.

Return values

- kStatus_MCG_ModeUnreachable – Could not switch to the target mode.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_SetFeeMode(uint8_t frdiv, *mcg_dmx32_t* dmx32, *mcg_drs_t* drs, void (*fllStableDelay)(void))

Sets the MCG to FEE mode.

This function sets the MCG to FEE mode. If setting to FEE mode fails from the current mode, this function returns an error.

Parameters

- frdiv – FLL reference clock divider setting, FRDIV.
- dmx32 – DMX32 in FEE mode.
- drs – The DCO range selection.
- fllStableDelay – Delay function to make sure FLL is stable. Passing NULL does not cause a delay.

Return values

- kStatus_MCG_ModeUnreachable – Could not switch to the target mode.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_SetFbiMode(*mcg_dmx32_t* dmx32, *mcg_drs_t* drs, void (*fllStableDelay)(void))

Sets the MCG to FBI mode.

This function sets the MCG to FBI mode. If setting to FBI mode fails from the current mode, this function returns an error.

Note: If dmx32 is set to kMCG_Dmx32Fine, the slow IRC must not be trimmed to frequency above 32768 Hz.

Parameters

- dmx32 – DMX32 in FBI mode.
- drs – The DCO range selection.
- fllStableDelay – Delay function to make sure FLL is stable. If the FLL is not used in FBI mode, this parameter can be NULL. Passing NULL does not cause a delay.

Return values

- kStatus_MCG_ModeUnreachable – Could not switch to the target mode.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_SetFbeMode(uint8_t frdiv, *mcg_dmx32_t* dmx32, *mcg_drs_t* drs, void (*fllStableDelay)(void))

Sets the MCG to FBE mode.

This function sets the MCG to FBE mode. If setting to FBE mode fails from the current mode, this function returns an error.

Parameters

- frdiv – FLL reference clock divider setting, FRDIV.
- dmx32 – DMX32 in FBE mode.
- drs – The DCO range selection.
- fllStableDelay – Delay function to make sure FLL is stable. If the FLL is not used in FBE mode, this parameter can be NULL. Passing NULL does not cause a delay.

Return values

- kStatus_MCG_ModeUnreachable – Could not switch to the target mode.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_SetBlpiMode(void)

Sets the MCG to BLPI mode.

This function sets the MCG to BLPI mode. If setting to BLPI mode fails from the current mode, this function returns an error.

Return values

- kStatus_MCG_ModeUnreachable – Could not switch to the target mode.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_SetBlpeMode(void)

Sets the MCG to BLPE mode.

This function sets the MCG to BLPE mode. If setting to BLPE mode fails from the current mode, this function returns an error.

Return values

- kStatus_MCG_ModeUnreachable – Could not switch to the target mode.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_SetPbeMode(*mcg_pll_clk_select_t* pllcs, *mcg_pll_config_t* const *config)

Sets the MCG to PBE mode.

This function sets the MCG to PBE mode. If setting to PBE mode fails from the current mode, this function returns an error.

Note:

a. The parameter pllcs selects the PLL. For platforms with only one PLL, the parameter pllcs is kept for interface compatibility.

b. The parameter config is the PLL configuration structure. On some platforms, it is possible to choose the external PLL directly, which renders the configuration structure not necessary. In this case, pass in NULL. For example: CLOCK_SetPbeMode(kMCG_OscselOsc, kMCG_PllClkSelExtPll, NULL);

Parameters

- pllcs – The PLL selection, PLLCS.

- config – Pointer to the PLL configuration.

Return values

- kStatus_MCG_ModeUnreachable – Could not switch to the target mode.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_SetPeeMode(void)

Sets the MCG to PEE mode.

This function sets the MCG to PEE mode.

Note: This function only changes the CLKS to use the PLL/FLL output. If the PRDIV/VDIV are different than in the PBE mode, set them up in PBE mode and wait. When the clock is stable, switch to PEE mode.

Return values

- kStatus_MCG_ModeUnreachable – Could not switch to the target mode.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_ExternalModeToFbeModeQuick(void)

Switches the MCG to FBE mode from the external mode.

This function switches the MCG from external modes (PEE/PBE/BLPE/FEE) to the FBE mode quickly. The external clock is used as the system clock source and PLL is disabled. However, the FLL settings are not configured. This is a lite function with a small code size, which is useful during the mode switch. For example, to switch from PEE mode to FEI mode:

```
CLOCK_ExternalModeToFbeModeQuick();
CLOCK_SetFeiMode(...);
```

Return values

- kStatus_Success – Switched successfully.
- kStatus_MCG_ModeInvalid – If the current mode is not an external mode, do not call this function.

status_t CLOCK_InternalModeToFbiModeQuick(void)

Switches the MCG to FBI mode from internal modes.

This function switches the MCG from internal modes (PEI/PBI/BLPI/FEI) to the FBI mode quickly. The MCGIRCLK is used as the system clock source and PLL is disabled. However, FLL settings are not configured. This is a lite function with a small code size, which is useful during the mode switch. For example, to switch from PEI mode to FEE mode:

```
CLOCK_InternalModeToFbiModeQuick();
CLOCK_SetFeeMode(...);
```

Return values

- kStatus_Success – Switched successfully.
- kStatus_MCG_ModeInvalid – If the current mode is not an internal mode, do not call this function.

status_t CLOCK_BootToFeiMode(*mcg_dmx32_t* dmx32, *mcg_drs_t* drs, void (*fllStableDelay)(void))

Sets the MCG to FEI mode during system boot up.

This function sets the MCG to FEI mode from the reset mode. It can also be used to set up MCG during system boot up.

Note: If dmx32 is set to kMCG_Dmx32Fine, the slow IRC must not be trimmed to frequency above 32768 Hz.

Parameters

- dmx32 – DMX32 in FEI mode.
- drs – The DCO range selection.
- fllStableDelay – Delay function to ensure that the FLL is stable.

Return values

- kStatus_MCG_ModeUnreachable – Could not switch to the target mode.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_BootToFeeMode(*mcg_oscsel_t* oscsel, uint8_t frdiv, *mcg_dmx32_t* dmx32, *mcg_drs_t* drs, void (*fllStableDelay)(void))

Sets the MCG to FEE mode during system bootup.

This function sets MCG to FEE mode from the reset mode. It can also be used to set up the MCG during system boot up.

Parameters

- oscsel – OSC clock select, OSCSEL.
- frdiv – FLL reference clock divider setting, FRDIV.
- dmx32 – DMX32 in FEE mode.
- drs – The DCO range selection.
- fllStableDelay – Delay function to ensure that the FLL is stable.

Return values

- kStatus_MCG_ModeUnreachable – Could not switch to the target mode.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_BootToBlpiMode(uint8_t fcrdiv, *mcg_irc_mode_t* ircs, uint8_t ircEnableMode)

Sets the MCG to BLPI mode during system boot up.

This function sets the MCG to BLPI mode from the reset mode. It can also be used to set up the MCG during system boot up.

Parameters

- fcrdiv – Fast IRC divider, FCRDIV.
- ircs – The internal reference clock to select, IRCS.
- ircEnableMode – The MCGIRCLK enable mode, OR'ed value of the enumeration _mcg_irclk_enable_mode.

Return values

- kStatus_MCG_SourceUsed – Could not change MCGIRCLK setting.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_BootToBlpeMode(*mcg_oscsel_t* oscsel)

Sets the MCG to BLPE mode during system boot up.

This function sets the MCG to BLPE mode from the reset mode. It can also be used to set up the MCG during system boot up.

Parameters

- oscsel – OSC clock select, MCG_C7[OSCSEL].

Return values

- kStatus_MCG_ModeUnreachable – Could not switch to the target mode.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_BootToPeeMode(*mcg_oscsel_t* oscsel, *mcg_pll_clk_select_t* pllcs, *mcg_pll_config_t* const *config)

Sets the MCG to PEE mode during system boot up.

This function sets the MCG to PEE mode from reset mode. It can also be used to set up the MCG during system boot up.

Parameters

- oscsel – OSC clock select, MCG_C7[OSCSEL].
- pllcs – The PLL selection, PLLCS.
- config – Pointer to the PLL configuration.

Return values

- kStatus_MCG_ModeUnreachable – Could not switch to the target mode.
- kStatus_Success – Switched to the target mode successfully.

status_t CLOCK_SetMcgConfig(*mcg_config_t* const *config)

Sets the MCG to a target mode.

This function sets MCG to a target mode defined by the configuration structure. If switching to the target mode fails, this function chooses the correct path.

Note: If the external clock is used in the target mode, ensure that it is enabled. For example, if the OSC0 is used, set up OSC0 correctly before calling this function.

Parameters

- config – Pointer to the target MCG mode configuration structure.

Returns

Return kStatus_Success if switched successfully; Otherwise, it returns an error code _mcg_status.

uint8_t pllFllSel

PLL/FLL/IRC48M selection.

uint8_t er32kSrc

ERCLK32K source selection.

uint32_t clkdiv1

SIM_CLKDIV1.

uint8_t enableMode
: OSCERCLK enable mode. OR'ed value of *_oscer_enable_mode*.

uint8_t erclkDiv
: Divider for OSCERCLK.

uint32_t freq
: External clock frequency.

uint8_t capLoad
: Capacitor load setting.

osc_mode_t workMode
: OSC work mode setting.

oscer_config_t oscerConfig
: Configuration for OSCERCLK.

uint8_t enableMode
: Enable mode. OR'ed value of enumeration _mcg_pll_enable_mode.

uint8_t prdiv
: Reference divider PRDIV.

uint8_t vdiv
: VCO divider VDIV.

mcg_mode_t mcgMode
: MCG mode.

uint8_t irclkEnableMode
: MCGIRCLK enable mode.

mcg_irc_mode_t ircs
: Source, MCG_C2[IRCS].

uint8_t fcrdiv
: Divider, MCG_SC[FCRDIV].

uint8_t frdiv
: Divider MCG_C1[FRDIV].

mcg_drs_t drs
: DCO range MCG_C4[DRST_DRS].

mcg_dmx32_t dmx32
: MCG_C4[DMX32].

mcg_oscsel_t oscsel

OSC select MCG_C7[OSCSEL].

mcg_pll_config_t pll0Config

MCGPLL0CLK configuration.

MCG_CONFIG_CHECK_PARAM

Configures whether to check a parameter in a function.

Some MCG settings must be changed with conditions, for example:

a. MCGIRCLK settings, such as the source, divider, and the trim value should not change when MCGIRCLK is used as a system clock source.

b. MCG_C7[OSCSEL] should not be changed when the external reference clock is used as a system clock source. For example, in FBE/BLPE/PBE modes.

c. The users should only switch between the supported clock modes.

MCG functions check the parameter and MCG status before setting, if not allowed to change, the functions return error. The parameter checking increases code size, if code size is a critical requirement, change *MCG_CONFIG_CHECK_PARAM* to 0 to disable parameter checking.

FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL

Configure whether driver controls clock.

When set to 0, peripheral drivers will enable clock in initialize function and disable clock in de-initialize function. When set to 1, peripheral driver will not control the clock, application could control the clock out of the driver.

Note: All drivers share this feature switcher. If it is set to 1, application should handle clock enable and disable for all drivers.

---

struct _sim_clock_config

#include <fsl_clock.h> SIM configuration structure for clock setting.

struct _oscer_config

#include <fsl_clock.h> OSC configuration for OSCERCLK.

struct _osc_config

#include <fsl_clock.h> OSC Initialization Configuration Structure.

Defines the configuration data structure to initialize the OSC. When porting to a new board, set the following members according to the board setting:

a. freq: The external frequency.

b. workMode: The OSC module mode.

struct _mcg_pll_config

#include <fsl_clock.h> MCG PLL configuration.

struct _mcg_config

#include <fsl_clock.h> MCG mode change configuration structure.

When porting to a new board, set the following members according to the board setting:

a. frdiv: If the FLL uses the external reference clock, set this value to ensure that the external reference clock divided by frdiv is in the 31.25 kHz to 39.0625 kHz range.

b. The PLL reference clock divider PRDIV: PLL reference clock frequency after PRDIV should be in the FSL_FEATURE_MCG_PLL_REF_MIN to FSL_FEATURE_MCG_PLL_REF_MAX range.

2.3 CMP: Analog Comparator Driver

void CMP_Init(CMP_Type *base, const *cmp_config_t* *config)

Initializes the CMP.

This function initializes the CMP module. The operations included are as follows.

- Enabling the clock for CMP module.
- Configuring the comparator.
- Enabling the CMP module. Note that for some devices, multiple CMP instances share the same clock gate. In this case, to enable the clock for any instance enables all CMPs. See the appropriate MCU reference manual for the clock assignment of the CMP.

Parameters

- base – CMP peripheral base address.
- config – Pointer to the configuration structure.

void CMP_Deinit(CMP_Type *base)

De-initializes the CMP module.

This function de-initializes the CMP module. The operations included are as follows.

- Disabling the CMP module.
- Disabling the clock for CMP module.

This function disables the clock for the CMP. Note that for some devices, multiple CMP instances share the same clock gate. In this case, before disabling the clock for the CMP, ensure that all the CMP instances are not used.

Parameters

- base – CMP peripheral base address.

static inline void CMP_Enable(CMP_Type *base, bool enable)

Enables/disables the CMP module.

Parameters

- base – CMP peripheral base address.
- enable – Enables or disables the module.

void CMP_GetDefaultConfig(*cmp_config_t* *config)

Initializes the CMP user configuration structure.

This function initializes the user configuration structure to these default values.

```
config->enableCmp            = true;
config->hysteresisMode       = kCMP_HysteresisLevel0;
config->enableHighSpeed      = false;
config->enableInvertOutput   = false;
config->useUnfilteredOutput  = false;
```

(continues on next page)

(continued from previous page)

```
config->enablePinOut        = false;
config->enableTriggerMode   = false;
```

Parameters

- config – Pointer to the configuration structure.

void CMP_SetInputChannels(CMP_Type *base, uint8_t positiveChannel, uint8_t negativeChannel)

Sets the input channels for the comparator.

This function sets the input channels for the comparator. Note that two input channels cannot be set the same way in the application. When the user selects the same input from the analog mux to the positive and negative port, the comparator is disabled automatically.

Parameters

- base – CMP peripheral base address.
- positiveChannel – Positive side input channel number. Available range is 0-7.
- negativeChannel – Negative side input channel number. Available range is 0-7.

void CMP_EnableDMA(CMP_Type *base, bool enable)

Enables/disables the DMA request for rising/falling events.

This function enables/disables the DMA request for rising/falling events. Either event triggers the generation of the DMA request from CMP if the DMA feature is enabled. Both events are ignored for generating the DMA request from the CMP if the DMA is disabled.

Parameters

- base – CMP peripheral base address.
- enable – Enables or disables the feature.

static inline void CMP_EnableWindowMode(CMP_Type *base, bool enable)

Enables/disables the window mode.

Parameters

- base – CMP peripheral base address.
- enable – Enables or disables the feature.

static inline void CMP_EnablePassThroughMode(CMP_Type *base, bool enable)

Enables/disables the pass through mode.

Parameters

- base – CMP peripheral base address.
- enable – Enables or disables the feature.

void CMP_SetFilterConfig(CMP_Type *base, const *cmp_filter_config_t* *config)

Configures the filter.

Parameters

- base – CMP peripheral base address.
- config – Pointer to the configuration structure.

void CMP_SetDACConfig(CMP_Type *base, const *cmp_dac_config_t* *config)

Configures the internal DAC.

Parameters

- base – CMP peripheral base address.
- config – Pointer to the configuration structure. "NULL" disables the feature.

void CMP_EnableInterrupts(CMP_Type *base, uint32_t mask)

Enables the interrupts.

Parameters

- base – CMP peripheral base address.
- mask – Mask value for interrupts. See "_cmp_interrupt_enable".

void CMP_DisableInterrupts(CMP_Type *base, uint32_t mask)

Disables the interrupts.

Parameters

- base – CMP peripheral base address.
- mask – Mask value for interrupts. See "_cmp_interrupt_enable".

uint32_t CMP_GetStatusFlags(CMP_Type *base)

Gets the status flags.

Parameters

- base – CMP peripheral base address.

Returns

Mask value for the asserted flags. See "_cmp_status_flags".

void CMP_ClearStatusFlags(CMP_Type *base, uint32_t mask)

Clears the status flags.

Parameters

- base – CMP peripheral base address.
- mask – Mask value for the flags. See "_cmp_status_flags".

FSL_CMP_DRIVER_VERSION

CMP driver version 2.0.3.

enum _cmp_interrupt_enable

Interrupt enable/disable mask.

Values:

enumerator kCMP_OutputRisingInterruptEnable

Comparator interrupt enable rising.

enumerator kCMP_OutputFallingInterruptEnable

Comparator interrupt enable falling.

enum _cmp_status_flags

Status flags' mask.

Values:

enumerator kCMP_OutputRisingEventFlag
Rising-edge on the comparison output has occurred.

enumerator kCMP_OutputFallingEventFlag
Falling-edge on the comparison output has occurred.

enumerator kCMP_OutputAssertEventFlag
Return the current value of the analog comparator output.

enum _cmp_hysteresis_mode
CMP Hysteresis mode.
Values:

enumerator kCMP_HysteresisLevel0
Hysteresis level 0.

enumerator kCMP_HysteresisLevel1
Hysteresis level 1.

enumerator kCMP_HysteresisLevel2
Hysteresis level 2.

enumerator kCMP_HysteresisLevel3
Hysteresis level 3.

enum _cmp_reference_voltage_source
CMP Voltage Reference source.
Values:

enumerator kCMP_VrefSourceVin1
Vin1 is selected as a resistor ladder network supply reference Vin.

enumerator kCMP_VrefSourceVin2
Vin2 is selected as a resistor ladder network supply reference Vin.

typedef enum *_cmp_hysteresis_mode* cmp_hysteresis_mode_t
CMP Hysteresis mode.

typedef enum *_cmp_reference_voltage_source* cmp_reference_voltage_source_t
CMP Voltage Reference source.

typedef struct *_cmp_config* cmp_config_t
Configures the comparator.

typedef struct *_cmp_filter_config* cmp_filter_config_t
Configures the filter.

typedef struct *_cmp_dac_config* cmp_dac_config_t

Configures the internal DAC.

struct _cmp_config

#include <fsl_cmp.h> Configures the comparator.

Public Members

bool enableCmp

Enable the CMP module.

cmp_hysteresis_mode_t hysteresisMode

CMP Hysteresis mode.

bool enableHighSpeed

Enable High-speed (HS) comparison mode.

bool enableInvertOutput

Enable the inverted comparator output.

bool useUnfilteredOutput

Set the compare output(COUT) to equal COUTA(true) or COUT(false).

bool enablePinOut

The comparator output is available on the associated pin.

bool enableTriggerMode

Enable the trigger mode.

struct _cmp_filter_config

#include <fsl_cmp.h> Configures the filter.

Public Members

bool enableSample

Using the external SAMPLE as a sampling clock input or using a divided bus clock.

uint8_t filterCount

Filter Sample Count. Available range is 1-7; 0 disables the filter.

uint8_t filterPeriod

Filter Sample Period. The divider to the bus clock. Available range is 0-255.

struct _cmp_dac_config

#include <fsl_cmp.h> Configures the internal DAC.

Public Members

cmp_reference_voltage_source_t referenceVoltageSource
Supply voltage reference source.

uint8_t DACValue
Value for the DAC Output Voltage. Available range is 0-63.

2.4 CRC: Cyclic Redundancy Check Driver

FSL_CRC_DRIVER_VERSION
CRC driver version. Version 2.1.0.

Current version: 2.1.0

Change log:

- Version 2.1.0
  - Choosing CRC clocks from CRC clock array according to instance instead of hard-coded value.
- Version 2.0.5
  - Fix CERT-C issue with boolean-to-unsigned integer conversion.
- Version 2.0.4
  - Release peripheral from reset if necessary in init function.
- Version 2.0.3
  - Fix MISRA issues
- Version 2.0.2
  - Fix MISRA issues
- Version 2.0.1
  - move DATA and DATALL macro definition from header file to source file

enum _crc_bits
CRC bit width.

Values:

enumerator kCrcBits16
Generate 16-bit CRC code

enumerator kCrcBits32
Generate 32-bit CRC code

enum _crc_result
CRC result type.

Values:

enumerator kCrcFinalChecksum

CRC data register read value is the final checksum. Reflect out and final xor protocol features are applied.

enumerator kCrcIntermediateChecksum

CRC data register read value is intermediate checksum (raw value). Reflect out and final xor protocol feature are not applied. Intermediate checksum can be used as a seed for *CRC_Init()* to continue adding data to this checksum.

typedef enum *_crc_bits* crc_bits_t

CRC bit width.

typedef enum *_crc_result* crc_result_t

CRC result type.

typedef struct *_crc_config* crc_config_t

CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

void CRC_Init(CRC_Type *base, const *crc_config_t* *config)

Enables and configures the CRC peripheral module.

This function enables the clock gate in the SIM module for the CRC peripheral. It also configures the CRC module and starts a checksum computation by writing the seed.

Parameters

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

void CRC_Deinit(CRC_Type *base)

Disables the CRC peripheral module.

This function disables the clock gate in the SIM module for the CRC peripheral.

Parameters

- base – CRC peripheral address.

void CRC_GetDefaultConfig(*crc_config_t* *config)

Loads default values to the CRC protocol configuration structure.

Loads default values to the CRC protocol configuration structure. The default values are as follows.

```
config->polynomial = 0x1021;
config->seed = 0xFFFF;
config->reflectIn = false;
config->reflectOut = false;
config->complementChecksum = false;
config->crcBits = kCrcBits16;
config->crcResult = kCrcFinalChecksum;
```

Parameters

- config – CRC protocol configuration structure.

void CRC_WriteData(CRC_Type *base, const uint8_t *data, size_t dataSize)

Writes data to the CRC module.

Writes input data buffer bytes to the CRC data register. The configured type of transpose is applied.

Parameters

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

uint32_t CRC_Get32bitResult(CRC_Type *base)

Reads the 32-bit checksum from the CRC module.

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

- base – CRC peripheral address.

Returns

An intermediate or the final 32-bit checksum, after configured transpose and complement operations.

uint16_t CRC_Get16bitResult(CRC_Type *base)

Reads a 16-bit checksum from the CRC module.

Reads the CRC data register (either an intermediate or the final checksum). The configured type of transpose and complement is applied.

Parameters

- base – CRC peripheral address.

Returns

An intermediate or the final 16-bit checksum, after configured transpose and complement operations.

CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT

Default configuration structure filled by *CRC_GetDefaultConfig()*. Use CRC16-CCIT-FALSE as defeault.

struct _crc_config

#include <fsl_crc.h> CRC protocol configuration.

This structure holds the configuration for the CRC protocol.

Public Members

uint32_t polynomial

CRC Polynomial, MSBit first. Example polynomial: 0x1021 = 1_0000_0010_0001 = x^12+x^5+1

uint32_t seed

Starting checksum value

bool reflectIn
: Reflect bits on input.

bool reflectOut
: Reflect bits on output.

bool complementChecksum
: True if the result shall be complement of the actual checksum.

crc_bits_t crcBits
: Selects 16- or 32- bit CRC protocol.

crc_result_t crcResult
: Selects final or intermediate checksum return from *CRC_Get16bitResult()* or *CRC_Get32bitResult()*

2.5 DAC: Digital-to-Analog Converter Driver

void DAC_Init(DAC_Type *base, const *dac_config_t* *config)
Initializes the DAC module.

This function initializes the DAC module including the following operations.

- Enabling the clock for DAC module.
- Configuring the DAC converter with a user configuration.
- Enabling the DAC module.

Parameters

- base – DAC peripheral base address.
- config – Pointer to the configuration structure. See "dac_config_t".

void DAC_Deinit(DAC_Type *base)
De-initializes the DAC module.

This function de-initializes the DAC module including the following operations.

- Disabling the DAC module.
- Disabling the clock for the DAC module.

Parameters

- base – DAC peripheral base address.

void DAC_GetDefaultConfig(*dac_config_t* *config)
Initializes the DAC user configuration structure.

This function initializes the user configuration structure to a default value. The default values are as follows.

```
config->referenceVoltageSource = kDAC_ReferenceVoltageSourceVref2;
config->enableLowPowerMode = false;
```

Parameters

- config – Pointer to the configuration structure. See “dac_config_t”.

static inline void DAC_Enable(DAC_Type *base, bool enable)

Enables the DAC module.

Parameters

- base – DAC peripheral base address.
- enable – Enables or disables the feature.

static inline void DAC_EnableBuffer(DAC_Type *base, bool enable)

Enables the DAC buffer.

Parameters

- base – DAC peripheral base address.
- enable – Enables or disables the feature.

void DAC_SetBufferConfig(DAC_Type *base, const *dac_buffer_config_t* *config)

Configures the CMP buffer.

Parameters

- base – DAC peripheral base address.
- config – Pointer to the configuration structure. See “dac_buffer_config_t”.

void DAC_GetDefaultBufferConfig(*dac_buffer_config_t* *config)

Initializes the DAC buffer configuration structure.

This function initializes the DAC buffer configuration structure to default values. The default values are as follows.

```
config->triggerMode = kDAC_BufferTriggerBySoftwareMode;
config->watermark   = kDAC_BufferWatermark1Word;
config->workMode    = kDAC_BufferWorkAsNormalMode;
config->upperLimit  = DAC_DATL_COUNT - 1U;
```

Parameters

- config – Pointer to the configuration structure. See “dac_buffer_config_t”.

static inline void DAC_EnableBufferDMA(DAC_Type *base, bool enable)

Enables the DMA for DAC buffer.

Parameters

- base – DAC peripheral base address.
- enable – Enables or disables the feature.

void DAC_SetBufferValue(DAC_Type *base, uint8_t index, uint16_t value)

Sets the value for items in the buffer.

Parameters

- base – DAC peripheral base address.
- index – Setting the index for items in the buffer. The available index should not exceed the size of the DAC buffer.
- value – Setting the value for items in the buffer. 12-bits are available.

static inline void DAC_DoSoftwareTriggerBuffer(DAC_Type *base)

Triggers the buffer using software and updates the read pointer of the DAC buffer.

This function triggers the function using software. The read pointer of the DAC buffer is updated with one step after this function is called. Changing the read pointer depends on the buffer's work mode.

Parameters

- base – DAC peripheral base address.

static inline uint8_t DAC_GetBufferReadPointer(DAC_Type *base)

Gets the current read pointer of the DAC buffer.

This function gets the current read pointer of the DAC buffer. The current output value depends on the item indexed by the read pointer. It is updated either by a software trigger or a hardware trigger.

Parameters

- base – DAC peripheral base address.

Returns

The current read pointer of the DAC buffer.

void DAC_SetBufferReadPointer(DAC_Type *base, uint8_t index)

Sets the current read pointer of the DAC buffer.

This function sets the current read pointer of the DAC buffer. The current output value depends on the item indexed by the read pointer. It is updated either by a software trigger or a hardware trigger. After the read pointer changes, the DAC output value also changes.

Parameters

- base – DAC peripheral base address.
- index – Setting an index value for the pointer.

void DAC_EnableBufferInterrupts(DAC_Type *base, uint32_t mask)

Enables interrupts for the DAC buffer.

Parameters

- base – DAC peripheral base address.
- mask – Mask value for interrupts. See "_dac_buffer_interrupt_enable".

void DAC_DisableBufferInterrupts(DAC_Type *base, uint32_t mask)

Disables interrupts for the DAC buffer.

Parameters

- base – DAC peripheral base address.
- mask – Mask value for interrupts. See "_dac_buffer_interrupt_enable".

uint8_t DAC_GetBufferStatusFlags(DAC_Type *base)

Gets the flags of events for the DAC buffer.

Parameters

- base – DAC peripheral base address.

Returns

Mask value for the asserted flags. See "_dac_buffer_status_flags".

void DAC_ClearBufferStatusFlags(DAC_Type *base, uint32_t mask)

Clears the flags of events for the DAC buffer.

Parameters

- base – DAC peripheral base address.
- mask – Mask value for flags. See "_dac_buffer_status_flags_t".

FSL_DAC_DRIVER_VERSION
DAC driver version 2.0.2.

enum _dac_buffer_status_flags
DAC buffer flags.
Values:

enumerator kDAC_BufferWatermarkFlag
DAC Buffer Watermark Flag.

enumerator kDAC_BufferReadPointerTopPositionFlag
DAC Buffer Read Pointer Top Position Flag.

enumerator kDAC_BufferReadPointerBottomPositionFlag
DAC Buffer Read Pointer Bottom Position Flag.

enum _dac_buffer_interrupt_enable
DAC buffer interrupts.
Values:

enumerator kDAC_BufferWatermarkInterruptEnable
DAC Buffer Watermark Interrupt Enable.

enumerator kDAC_BufferReadPointerTopInterruptEnable
DAC Buffer Read Pointer Top Flag Interrupt Enable.

enumerator kDAC_BufferReadPointerBottomInterruptEnable
DAC Buffer Read Pointer Bottom Flag Interrupt Enable

enum _dac_reference_voltage_source
DAC reference voltage source.
Values:

enumerator kDAC_ReferenceVoltageSourceVref1
The DAC selects DACREF_1 as the reference voltage.

enumerator kDAC_ReferenceVoltageSourceVref2
The DAC selects DACREF_2 as the reference voltage.

enum _dac_buffer_trigger_mode
DAC buffer trigger mode.
Values:

enumerator kDAC_BufferTriggerByHardwareMode
The DAC hardware trigger is selected.

enumerator kDAC_BufferTriggerBySoftwareMode
The DAC software trigger is selected.

enum _dac_buffer_watermark
DAC buffer watermark.
Values:

enumerator kDAC_BufferWatermark1Word
1 word away from the upper limit.

enumerator kDAC_BufferWatermark2Word
2 words away from the upper limit.

enumerator kDAC_BufferWatermark3Word
3 words away from the upper limit.

enumerator kDAC_BufferWatermark4Word
4 words away from the upper limit.

enum _dac_buffer_work_mode
DAC buffer work mode.
Values:

enumerator kDAC_BufferWorkAsNormalMode
Normal mode.

enumerator kDAC_BufferWorkAsSwingMode
Swing mode.

enumerator kDAC_BufferWorkAsOneTimeScanMode
One-Time Scan mode.

enumerator kDAC_BufferWorkAsFIFOMode
FIFO mode.

typedef enum *_dac_reference_voltage_source* dac_reference_voltage_source_t
DAC reference voltage source.

typedef enum *_dac_buffer_trigger_mode* dac_buffer_trigger_mode_t
DAC buffer trigger mode.

typedef enum *_dac_buffer_watermark* dac_buffer_watermark_t
DAC buffer watermark.

typedef enum *_dac_buffer_work_mode* dac_buffer_work_mode_t
DAC buffer work mode.

typedef struct *_dac_config* dac_config_t

DAC module configuration.

typedef struct *_dac_buffer_config* dac_buffer_config_t

DAC buffer configuration.

struct _dac_config

#include <fsl_dac.h> DAC module configuration.

Public Members

dac_reference_voltage_source_t referenceVoltageSource

Select the DAC reference voltage source.

bool enableLowPowerMode

Enable the low-power mode.

struct _dac_buffer_config

#include <fsl_dac.h> DAC buffer configuration.

Public Members

dac_buffer_trigger_mode_t triggerMode

Select the buffer's trigger mode.

dac_buffer_watermark_t watermark

Select the buffer's watermark.

dac_buffer_work_mode_t workMode

Select the buffer's work mode.

uint8_t upperLimit

Set the upper limit for the buffer index. Normally, 0-15 is available for a buffer with 16 items.

2.6 DMAMUX: Direct Memory Access Multiplexer Driver

void DMAMUX_Init(DMAMUX_Type *base)

Initializes the DMAMUX peripheral.

This function ungates the DMAMUX clock.

Parameters

- base – DMAMUX peripheral base address.

void DMAMUX_Deinit(DMAMUX_Type *base)

Deinitializes the DMAMUX peripheral.

This function gates the DMAMUX clock.

Parameters

- base – DMAMUX peripheral base address.

static inline void DMAMUX_EnableChannel(DMAMUX_Type *base, uint32_t channel)

Enables the DMAMUX channel.

This function enables the DMAMUX channel.

Parameters

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

static inline void DMAMUX_DisableChannel(DMAMUX_Type *base, uint32_t channel)

Disables the DMAMUX channel.

This function disables the DMAMUX channel.

Note: The user must disable the DMAMUX channel before configuring it.

Parameters

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

static inline void DMAMUX_SetSource(DMAMUX_Type *base, uint32_t channel, int32_t source)

Configures the DMAMUX channel source.

Parameters

- base – DMAMUX peripheral base address.
- channel – DMAMUX channel number.
- source – Channel source, which is used to trigger the DMA transfer.User need to use the dma_request_source_t type as the input parameter.

static inline void DMAMUX_EnablePeriodTrigger(DMAMUX_Type *base, uint32_t channel)

Enables the DMAMUX period trigger.

This function enables the DMAMUX period trigger feature.

Parameters

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

static inline void DMAMUX_DisablePeriodTrigger(DMAMUX_Type *base, uint32_t channel)

Disables the DMAMUX period trigger.

This function disables the DMAMUX period trigger.

Parameters

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

FSL_DMAMUX_DRIVER_VERSION

DMAMUX driver version 2.1.4.

2.7 DSPI: Serial Peripheral Interface Driver

2.8 DSPI Driver

void DSPI_MasterInit(SPI_Type *base, const *dspi_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Initializes the DSPI master.

This function initializes the DSPI master configuration. This is an example use case.

```
dspi_master_config_t  masterConfig;
masterConfig.whichCtar                                = kDSPI_Ctar0;
masterConfig.ctarConfig.baudRate                      = 500000000U;
masterConfig.ctarConfig.bitsPerFrame                  = 8;
masterConfig.ctarConfig.cpol                          = kDSPI_ClockPolarityActiveHigh;
masterConfig.ctarConfig.cpha                          = kDSPI_ClockPhaseFirstEdge;
masterConfig.ctarConfig.direction                     = kDSPI_MsbFirst;
masterConfig.ctarConfig.pcsToSckDelayInNanoSec        = 1000000000U / masterConfig.ctarConfig.
↪baudRate ;
masterConfig.ctarConfig.lastSckToPcsDelayInNanoSec    = 1000000000U / masterConfig.ctarConfig.
↪baudRate ;
masterConfig.ctarConfig.betweenTransferDelayInNanoSec = 1000000000U / masterConfig.ctarConfig.
↪baudRate ;
masterConfig.whichPcs                                 = kDSPI_Pcs0;
masterConfig.pcsActiveHighOrLow                       = kDSPI_PcsActiveLow;
masterConfig.enableContinuousSCK                      = false;
masterConfig.enableRxFifoOverWrite                    = false;
masterConfig.enableModifiedTimingFormat               = false;
masterConfig.samplePoint                              = kDSPI_SckToSin0Clock;
DSPI_MasterInit(base, &masterConfig, srcClock_Hz);
```

Parameters

- base – DSPI peripheral address.
- masterConfig – Pointer to the structure *dspi_master_config_t*.
- srcClock_Hz – Module source input clock in Hertz.

void DSPI_MasterGetDefaultConfig(*dspi_master_config_t* *masterConfig)

Sets the *dspi_master_config_t* structure to default values.

The purpose of this API is to get the configuration structure initialized for the *DSPI_MasterInit()*. Users may use the initialized structure unchanged in the *DSPI_MasterInit()* or modify the structure before calling the *DSPI_MasterInit()*. Example:

```
dspi_master_config_t  masterConfig;
DSPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

- masterConfig – pointer to *dspi_master_config_t* structure

void DSPI_SlaveInit(SPI_Type *base, const *dspi_slave_config_t* *slaveConfig)

DSPI slave configuration.

This function initializes the DSPI slave configuration. This is an example use case.

```
dspi_slave_config_t slaveConfig;
slaveConfig->whichCtar                = kDSPI_Ctar0;
slaveConfig->ctarConfig.bitsPerFrame  = 8;
slaveConfig->ctarConfig.cpol          = kDSPI_ClockPolarityActiveHigh;
slaveConfig->ctarConfig.cpha          = kDSPI_ClockPhaseFirstEdge;
slaveConfig->enableContinuousSCK      = false;
slaveConfig->enableRxFifoOverWrite    = false;
slaveConfig->enableModifiedTimingFormat = false;
slaveConfig->samplePoint              = kDSPI_SckToSin0Clock;
DSPI_SlaveInit(base, &slaveConfig);
```

Parameters

- base – DSPI peripheral address.
- slaveConfig – Pointer to the structure *dspi_master_config_t*.

void DSPI_SlaveGetDefaultConfig(*dspi_slave_config_t* *slaveConfig)

Sets the *dspi_slave_config_t* structure to a default value.

The purpose of this API is to get the configuration structure initialized for the *DSPI_SlaveInit()*. Users may use the initialized structure unchanged in the *DSPI_SlaveInit()* or modify the structure before calling the *DSPI_SlaveInit()*. This is an example.

```
dspi_slave_config_t slaveConfig;
DSPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

- slaveConfig – Pointer to the *dspi_slave_config_t* structure.

void DSPI_Deinit(SPI_Type *base)

De-initializes the DSPI peripheral. Call this API to disable the DSPI clock.

Parameters

- base – DSPI peripheral address.

static inline void DSPI_Enable(SPI_Type *base, bool enable)

Enables the DSPI peripheral and sets the MCR MDIS to 0.

Parameters

- base – DSPI peripheral address.
- enable – Pass true to enable module, false to disable module.

static inline uint32_t DSPI_GetStatusFlags(SPI_Type *base)

Gets the DSPI status flag state.

Parameters

- base – DSPI peripheral address.

Returns

DSPI status (in SR register).

static inline void DSPI_ClearStatusFlags(SPI_Type *base, uint32_t statusFlags)

Clears the DSPI status flag.

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status bit to clear. The list of status bits is defined in the **dspi_status_and_interrupt_request_t**. The function uses these bit positions in its algorithm to clear the desired flag state. This is an example.

```
DSPI_ClearStatusFlags(base, kDSPI_TxCompleteFlag|kDSPI_EndOfQueueFlag);
```

Parameters

- base – DSPI peripheral address.
- statusFlags – The status flag used from the type dspi_flags.

void DSPI_EnableInterrupts(SPI_Type *base, uint32_t mask)

Enables the DSPI interrupts.

This function configures various interrupt masks of the DSPI. The parameters are a base and an interrupt mask.

```
DSPI_EnableInterrupts(base, kDSPI_TxCompleteInterruptEnable | kDSPI_
↪EndOfQueueInterruptEnable );
```

Note: For Tx Fill and Rx FIFO drain requests, enable the interrupt request and disable the DMA request. Do not use this API(write to RSER register) while DSPI is in running state.

Parameters

- base – DSPI peripheral address.
- mask – The interrupt mask; use the enum *_dspi_interrupt_enable*.

static inline void DSPI_DisableInterrupts(SPI_Type *base, uint32_t mask)

Disables the DSPI interrupts.

```
DSPI_DisableInterrupts(base, kDSPI_TxCompleteInterruptEnable | kDSPI_
↪EndOfQueueInterruptEnable );
```

Parameters

- base – DSPI peripheral address.
- mask – The interrupt mask; use the enum *_dspi_interrupt_enable*.

static inline void DSPI_EnableDMA(SPI_Type *base, uint32_t mask)

Enables the DSPI DMA request.

This function configures the Rx and Tx DMA mask of the DSPI. The parameters are a base and a DMA mask.

```
DSPI_EnableDMA(base, kDSPI_TxDmaEnable | kDSPI_RxDmaEnable);
```

Parameters

- base – DSPI peripheral address.
- mask – The interrupt mask; use the enum *_dspi_dma_enable*.

static inline void DSPI_DisableDMA(SPI_Type *base, uint32_t mask)

Disables the DSPI DMA request.

This function configures the Rx and Tx DMA mask of the DSPI. The parameters are a base and a DMA mask.

```
SPI_DisableDMA(base, kDSPI_TxDmaEnable | kDSPI_RxDmaEnable);
```

Parameters

- base – DSPI peripheral address.
- mask – The interrupt mask; use the enum *_dspi_dma_enable*.

static inline uint32_t DSPI_MasterGetTxRegisterAddress(SPI_Type *base)

Gets the DSPI master PUSHR data register address for the DMA operation.

This function gets the DSPI master PUSHR data register address because this value is needed for the DMA operation.

Parameters

- base – DSPI peripheral address.

Returns

The DSPI master PUSHR data register address.

static inline uint32_t DSPI_SlaveGetTxRegisterAddress(SPI_Type *base)

Gets the DSPI slave PUSHR data register address for the DMA operation.

This function gets the DSPI slave PUSHR data register address as this value is needed for the DMA operation.

Parameters

- base – DSPI peripheral address.

Returns

The DSPI slave PUSHR data register address.

static inline uint32_t DSPI_GetRxRegisterAddress(SPI_Type *base)

Gets the DSPI POPR data register address for the DMA operation.

This function gets the DSPI POPR data register address as this value is needed for the DMA operation.

Parameters

- base – DSPI peripheral address.

Returns

The DSPI POPR data register address.

uint32_t DSPI_GetInstance(SPI_Type *base)

Get instance number for DSPI module.

Parameters

- base – DSPI peripheral base address.

static inline void DSPI_SetMasterSlaveMode(SPI_Type *base, *dspi_master_slave_mode_t* mode)

Configures the DSPI for master or slave.

Parameters

- base – DSPI peripheral address.
- mode – Mode setting (master or slave) of type *dspi_master_slave_mode_t*.

static inline bool DSPI_IsMaster(SPI_Type *base)

Returns whether the DSPI module is in master mode.

Parameters

- base – DSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

static inline void DSPI_StartTransfer(SPI_Type *base)

Starts the DSPI transfers and clears HALT bit in MCR.

This function sets the module to start data transfer in either master or slave mode.

Parameters

- base – DSPI peripheral address.

static inline void DSPI_StopTransfer(SPI_Type *base)

Stops DSPI transfers and sets the HALT bit in MCR.

This function stops data transfers in either master or slave modes.

Parameters

- base – DSPI peripheral address.

static inline void DSPI_SetFifoEnable(SPI_Type *base, bool enableTxFifo, bool enableRxFifo)

Enables or disables the DSPI FIFOs.

This function allows the caller to disable/enable the Tx and Rx FIFOs independently.

Note: To disable, pass in a logic 0 (false) for the particular FIFO configuration. To enable, pass in a logic 1 (true).

Parameters

- base – DSPI peripheral address.
- enableTxFifo – Disables (false) the TX FIFO; Otherwise, enables (true) the TX FIFO
- enableRxFifo – Disables (false) the RX FIFO; Otherwise, enables (true) the RX FIFO

static inline void DSPI_FlushFifo(SPI_Type *base, bool flushTxFifo, bool flushRxFifo)

Flushes the DSPI FIFOs.

Parameters

- base – DSPI peripheral address.
- flushTxFifo – Flushes (true) the Tx FIFO; Otherwise, does not flush (false) the Tx FIFO
- flushRxFifo – Flushes (true) the Rx FIFO; Otherwise, does not flush (false) the Rx FIFO

static inline void DSPI_SetAllPcsPolarity(SPI_Type *base, uint32_t mask)

Configures the DSPI peripheral chip select polarity simultaneously. For example, PCS0 and PCS1 are set to active low and other PCS is set to active high. Note that the number of PCSs is specific to the device.

```
DSPI_SetAllPcsPolarity(base, kDSPI_Pcs0ActiveLow | kDSPI_Pcs1ActiveLow);
```

Parameters

- base – DSPI peripheral address.
- mask – The PCS polarity mask; use the enum *_dspi_pcs_polarity*.

uint32_t DSPI_MasterSetBaudRate(SPI_Type *base, *dspi_ctar_selection_t* whichCtar, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the DSPI baud rate in bits per second.

This function takes in the desired baudRate_Bps (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate, and returns the calculated baud rate in bits-per-second. It requires that the caller also provide the frequency of the module source clock (in Hertz).

Parameters

- base – DSPI peripheral address.
- whichCtar – The desired Clock and Transfer Attributes Register (CTAR) of the type *dspi_ctar_selection_t*
- baudRate_Bps – The desired baud rate in bits per second
- srcClock_Hz – Module source input clock in Hertz

Returns

The actual calculated baud rate

void DSPI_MasterSetDelayScaler(SPI_Type *base, *dspi_ctar_selection_t* whichCtar, uint32_t prescaler, uint32_t scaler, *dspi_delay_type_t* whichDelay)

Manually configures the delay prescaler and scaler for a particular CTAR.

This function configures the PCS to SCK delay pre-scalar (PcsSCK) and scalar (CSSCK), after SCK delay pre-scalar (PASC) and scalar (ASC), and the delay after transfer pre-scalar (PDT) and scalar (DT).

These delay names are available in the type *dspi_delay_type_t*.

The user passes the delay to the configuration along with the prescaler and scaler value. This allows the user to directly set the prescaler/scaler values if pre-calculated or to manually increment either value.

Parameters

- base – DSPI peripheral address.
- whichCtar – The desired Clock and Transfer Attributes Register (CTAR) of type *dspi_ctar_selection_t*.
- prescaler – The prescaler delay value (can be an integer 0, 1, 2, or 3).
- scaler – The scaler delay value (can be any integer between 0 to 15).
- whichDelay – The desired delay to configure; must be of type *dspi_delay_type_t*

uint32_t DSPI_MasterSetDelayTimes(SPI_Type *base, *dspi_ctar_selection_t* whichCtar, *dspi_delay_type_t* whichDelay, uint32_t srcClock_Hz, uint32_t delayTimeInNanoSec)

Calculates the delay prescaler and scaler based on the desired delay input in nanoseconds.

This function calculates the values for the following. PCS to SCK delay pre-scalar (PCSSCK) and scalar (CSSCK), or After SCK delay pre-scalar (PASC) and scalar (ASC), or Delay after transfer pre-scalar (PDT) and scalar (DT).

These delay names are available in the type *dspi_delay_type_t*.

The user passes which delay to configure along with the desired delay value in nanoseconds. The function calculates the values needed for the prescaler and scaler. Note that returning the calculated delay as an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. The higher-level peripheral driver alerts the user of an out of range delay input.

Parameters

- base – DSPI peripheral address.
- whichCtar – The desired Clock and Transfer Attributes Register (CTAR) of type *dspi_ctar_selection_t*.
- whichDelay – The desired delay to configure, must be of type *dspi_delay_type_t*
- srcClock_Hz – Module source input clock in Hertz
- delayTimeInNanoSec – The desired delay value in nanoseconds.

Returns

The actual calculated delay value.

static inline void DSPI_MasterWriteData(SPI_Type *base, *dspi_command_data_config_t* *command, uint16_t data)

Writes data into the data buffer for master mode.

In master mode, the 16-bit data is appended to the 16-bit command info. The command portion provides characteristics of the data, such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). This is an example.

```
dspi_command_data_config_t commandConfig;
commandConfig.isPcsContinuous = true;
commandConfig.whichCtar = kDSPICtar0;
commandConfig.whichPcs = kDSPIPcs0;
commandConfig.clearTransferCount = false;
commandConfig.isEndOfQueue = false;
DSPI_MasterWriteData(base, &commandConfig, dataWord);
```

Parameters

- base – DSPI peripheral address.
- command – Pointer to the command structure.
- data – The data word to be sent.

void DSPI_GetDefaultDataCommandConfig(*dspi_command_data_config_t* *command)

Sets the *dspi_command_data_config_t* structure to default values.

The purpose of this API is to get the configuration structure initialized for use in the **DSPI_MasterWrite_xx()**. Users may use the initialized structure unchanged in the DSPI_MasterWrite_xx() or modify the structure before calling the DSPI_MasterWrite_xx(). This is an example.

```
dspi_command_data_config_t  command;
DSPI_GetDefaultDataCommandConfig(&command);
```

Parameters

- command – Pointer to the *dspi_command_data_config_t* structure.

void DSPI_MasterWriteDataBlocking(SPI_Type *base, *dspi_command_data_config_t* *command, uint16_t data)

Writes data into the data buffer master mode and waits till complete to return.

In master mode, the 16-bit data is appended to the 16-bit command info. The command portion provides characteristics of the data, such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). This is an example.

```
dspi_command_config_t commandConfig;
commandConfig.isPcsContinuous = true;
commandConfig.whichCtar = kDSPICtar0;
commandConfig.whichPcs = kDSPIPcs1;
commandConfig.clearTransferCount = false;
commandConfig.isEndOfQueue = false;
DSPI_MasterWriteDataBlocking(base, &commandConfig, dataWord);
```

Note: This function does not return until after the transmit is complete. Also note that the DSPI must be enabled and running to transmit data (MCR[MDIS] & [HALT] = 0). Because the SPI is a synchronous protocol, the received data is available when the transmit completes.

Parameters

- base – DSPI peripheral address.
- command – Pointer to the command structure.
- data – The data word to be sent.

static inline uint32_t DSPI_MasterGetFormattedCommand(*dspi_command_data_config_t* *command)

Returns the DSPI command word formatted to the PUSHR data register bit field.

This function allows the caller to pass in the data command structure and returns the command word formatted according to the DSPI PUSHR register bit field placement. The user can then "OR" the returned command word with the desired data to send and use the function **DSPI_HAL_WriteCommandDataMastermode** or **DSPI_HAL_WriteCommandDataMastermodeBlocking** to write the entire 32-bit command data word to the PUSHR. This helps improve performance in cases where the command structure is constant. For example, the user calls this function before starting a transfer to generate the command word. When they are ready to transmit the data, they OR this formatted command word with the desired data to transmit. This process increases transmit performance when compared to calling send functions, such as **DSPI_HAL_WriteDataMastermode**, which format the command word each time a data word is to be sent.

Parameters

- command – Pointer to the command structure.

Returns

The command word formatted to the PUSHR data register bit field.

void DSPI_MasterWriteCommandDataBlocking(SPI_Type *base, uint32_t data)

Writes a 32-bit data word (16-bit command appended with 16-bit data) into the data buffer master mode and waits till complete to return.

In this function, the user must append the 16-bit data to the 16-bit command information and then provide the total 32-bit word as the data to send. The command portion provides characteristics of the data, such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). The user is responsible for appending this command with the data to send. This is an example:

```
dataWord = <16-bit command> | <16-bit data>;
DSPI_MasterWriteCommandDataBlocking(base, dataWord);
```

For a blocking polling transfer, see methods below.

Option 1
uint32_t command_to_send = DSPI_MasterGetFormattedCommand(&command);
uint32_t data0 = command_to_send | data_need_to_send_0;
uint32_t data1 = command_to_send | data_need_to_send_1;
uint32_t data2 = command_to_send | data_need_to_send_2;
DSPI_MasterWriteCommandDataBlocking(base,data0);
DSPI_MasterWriteCommandDataBlocking(base,data1);
DSPI_MasterWriteCommandDataBlocking(base,data2);

Option 2
DSPI_MasterWriteDataBlocking(base,&command,data_need_to_send_0);
DSPI_MasterWriteDataBlocking(base,&command,data_need_to_send_1);
DSPI_MasterWriteDataBlocking(base,&command,data_need_to_send_2);

Note: This function does not return until after the transmit is complete. Also note that the DSPI must be enabled and running to transmit data (MCR[MDIS] & [HALT] = 0). Because the SPI is a synchronous protocol, the received data is available when the transmit completes.

Parameters

- base – DSPI peripheral address.
- data – The data word (command and data combined) to be sent.

static inline void DSPI_SlaveWriteData(SPI_Type *base, uint32_t data)

Writes data into the data buffer in slave mode.

In slave mode, up to 16-bit words may be written.

Parameters

- base – DSPI peripheral address.

- data – The data to send.

void DSPI_SlaveWriteDataBlocking(SPI_Type *base, uint32_t data)

Writes data into the data buffer in slave mode, waits till data was transmitted, and returns.

In slave mode, up to 16-bit words may be written. The function first clears the transmit complete flag, writes data into data register, and finally waits until the data is transmitted.

Parameters

- base – DSPI peripheral address.
- data – The data to send.

static inline uint32_t DSPI_ReadData(SPI_Type *base)

Reads data from the data buffer.

Parameters

- base – DSPI peripheral address.

Returns

The data from the read data buffer.

void DSPI_SetDummyData(SPI_Type *base, uint8_t dummyData)

Set up the dummy data.

Parameters

- base – DSPI peripheral address.
- dummyData – Data to be transferred when tx buffer is NULL.

void DSPI_MasterTransferCreateHandle(SPI_Type *base, *dspi_master_handle_t* *handle, *dspi_master_transfer_callback_t* callback, void *userData)

Initializes the DSPI master handle.

This function initializes the DSPI handle, which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Parameters

- base – DSPI peripheral base address.
- handle – DSPI handle pointer to *_dspi_master_handle*.
- callback – DSPI callback.
- userData – Callback function parameter.

status_t DSPI_MasterTransferBlocking(SPI_Type *base, *dspi_transfer_t* *transfer)

DSPI master transfer data using polling.

This function transfers data using polling. This is a blocking function, which does not return until all transfers have been completed.

Parameters

- base – DSPI peripheral base address.
- transfer – Pointer to the *dspi_transfer_t* structure.

Returns

status of status_t.

status_t DSPI_MasterTransferNonBlocking(SPI_Type *base, *dspi_master_handle_t* *handle, *dspi_transfer_t* *transfer)

DSPI master transfer data using interrupts.

This function transfers data using interrupts. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

- base – DSPI peripheral base address.
- handle – Pointer to the *_dspi_master_handle* structure which stores the transfer state.
- transfer – Pointer to the *dspi_transfer_t* structure.

Returns
status of status_t.

status_t DSPI_MasterHalfDuplexTransferBlocking(SPI_Type *base, *dspi_half_duplex_transfer_t* *xfer)

Transfers a block of data using a polling method.

This function will do a half-duplex transfer for DSPI master, This is a blocking function, which does not retuen until all transfer have been completed. And data transfer will be half-duplex, users can set transmit first or receive first.

Parameters

- base – DSPI base pointer
- xfer – pointer to *dspi_half_duplex_transfer_t* structure

Returns
status of status_t.

status_t DSPI_MasterHalfDuplexTransferNonBlocking(SPI_Type *base, *dspi_master_handle_t* *handle, *dspi_half_duplex_transfer_t* *xfer)

Performs a non-blocking DSPI interrupt transfer.

This function transfers data using interrupts, the transfer mechanism is half-duplex. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

- base – DSPI peripheral base address.
- handle – pointer to *_dspi_master_handle* structure which stores the transfer state
- xfer – pointer to *dspi_half_duplex_transfer_t* structure

Returns
status of status_t.

status_t DSPI_MasterTransferGetCount(SPI_Type *base, *dspi_master_handle_t* *handle, size_t *count)

Gets the master transfer count.

This function gets the master transfer count.

Parameters

- base – DSPI peripheral base address.
- handle – Pointer to the *_dspi_master_handle* structure which stores the transfer state.

- count – The number of bytes transferred by using the non-blocking transaction.

Returns
status of status_t.

void DSPI_MasterTransferAbort(SPI_Type *base, *dspi_master_handle_t* *handle)

DSPI master aborts a transfer using an interrupt.

This function aborts a transfer using an interrupt.

Parameters

- base – DSPI peripheral base address.
- handle – Pointer to the *_dspi_master_handle* structure which stores the transfer state.

void DSPI_MasterTransferHandleIRQ(SPI_Type *base, *dspi_master_handle_t* *handle)

DSPI Master IRQ handler function.

This function processes the DSPI transmit and receive IRQ.

Parameters

- base – DSPI peripheral base address.
- handle – Pointer to the *_dspi_master_handle* structure which stores the transfer state.

void DSPI_SlaveTransferCreateHandle(SPI_Type *base, *dspi_slave_handle_t* *handle, *dspi_slave_transfer_callback_t* callback, void *userData)

Initializes the DSPI slave handle.

This function initializes the DSPI handle, which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Parameters

- handle – DSPI handle pointer to the *_dspi_slave_handle*.
- base – DSPI peripheral base address.
- callback – DSPI callback.
- userData – Callback function parameter.

status_t DSPI_SlaveTransferNonBlocking(SPI_Type *base, *dspi_slave_handle_t* *handle, *dspi_transfer_t* *transfer)

DSPI slave transfers data using an interrupt.

This function transfers data using an interrupt. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

- base – DSPI peripheral base address.
- handle – Pointer to the *_dspi_slave_handle* structure which stores the transfer state.
- transfer – Pointer to the *dspi_transfer_t* structure.

Returns
status of status_t.

status_t DSPI_SlaveTransferGetCount(SPI_Type *base, *dspi_slave_handle_t* *handle, size_t *count)

Gets the slave transfer count.

This function gets the slave transfer count.

Parameters

- base – DSPI peripheral base address.
- handle – Pointer to the *_dspi_master_handle* structure which stores the transfer state.
- count – The number of bytes transferred by using the non-blocking transaction.

Returns

status of status_t.

void DSPI_SlaveTransferAbort(SPI_Type *base, *dspi_slave_handle_t* *handle)

DSPI slave aborts a transfer using an interrupt.

This function aborts a transfer using an interrupt.

Parameters

- base – DSPI peripheral base address.
- handle – Pointer to the *_dspi_slave_handle* structure which stores the transfer state.

void DSPI_SlaveTransferHandleIRQ(SPI_Type *base, *dspi_slave_handle_t* *handle)

DSPI Master IRQ handler function.

This function processes the DSPI transmit and receive IRQ.

Parameters

- base – DSPI peripheral base address.
- handle – Pointer to the *_dspi_slave_handle* structure which stores the transfer state.

uint8_t DSPI_GetDummyDataInstance(SPI_Type *base)

brief Dummy data for each instance.

The purpose of this API is to avoid MISRA rule8.5 : Multiple declarations of externally-linked object or function *g_dspiDummyData*.

param base DSPI peripheral base address.

void DSPI_DriverIRQHandler(uint32_t instance)

Common IRQ handler for DSPI, which can be used for DSPI IRQ remapping.

Parameters

- instance – DSPI instance number.

FSL_DSPI_DRIVER_VERSION

DSPI driver version 2.2.8.

Status for the DSPI driver.

Values:

enumerator kStatus_DSPI_Busy
DSPI transfer is busy.

enumerator kStatus_DSPI_Error
DSPI driver error.

enumerator kStatus_DSPI_Idle
DSPI is idle.

enumerator kStatus_DSPI_OutOfRange
DSPI transfer out of range.

enum _dspi_flags
DSPI status flags in SPIx_SR register.
Values:

enumerator kDSPI_TxCompleteFlag
Transfer Complete Flag.

enumerator kDSPI_EndOfQueueFlag
End of Queue Flag.

enumerator kDSPI_TxFifoUnderflowFlag
Transmit FIFO Underflow Flag.

enumerator kDSPI_TxFifoFillRequestFlag
Transmit FIFO Fill Flag.

enumerator kDSPI_RxFifoOverflowFlag
Receive FIFO Overflow Flag.

enumerator kDSPI_RxFifoDrainRequestFlag
Receive FIFO Drain Flag.

enumerator kDSPI_TxAndRxStatusFlag
The module is in Stopped/Running state.

enumerator kDSPI_AllStatusFlag
All statuses above.

enum _dspi_interrupt_enable
DSPI interrupt source.
Values:

enumerator kDSPI_TxCompleteInterruptEnable
TCF interrupt enable.

enumerator kDSPI_EndOfQueueInterruptEnable
EOQF interrupt enable.

enumerator kDSPI_TxFifoUnderflowInterruptEnable
TFUF interrupt enable.

enumerator kDSPI_TxFifoFillRequestInterruptEnable
TFFF interrupt enable, DMA disable.

enumerator kDSPI_RxFifoOverflowInterruptEnable
RFOF interrupt enable.

enumerator kDSPI_RxFifoDrainRequestInterruptEnable
RFDF interrupt enable, DMA disable.

enumerator kDSPI_AllInterruptEnable
All above interrupts enable.

enum _dspi_dma_enable
DSPI DMA source.
Values:

enumerator kDSPI_TxDmaEnable
TFFF flag generates DMA requests. No Tx interrupt request.

enumerator kDSPI_RxDmaEnable
RFDF flag generates DMA requests. No Rx interrupt request.

enum _dspi_master_slave_mode
DSPI master or slave mode configuration.
Values:

enumerator kDSPI_Master
DSPI peripheral operates in master mode.

enumerator kDSPI_Slave
DSPI peripheral operates in slave mode.

enum _dspi_master_sample_point
DSPI Sample Point: Controls when the DSPI master samples SIN in the Modified Transfer Format. This field is valid only when the CPHA bit in the CTAR register is 0.
Values:

enumerator kDSPI_SckToSin0Clock
0 system clocks between SCK edge and SIN sample.

enumerator kDSPI_SckToSin1Clock
1 system clock between SCK edge and SIN sample.

enumerator kDSPI_SckToSin2Clock
2 system clocks between SCK edge and SIN sample.

enum _dspi_which_pcs_config
DSPI Peripheral Chip Select (Pcs) configuration (which Pcs to configure).
Values:

enumerator kDSPI_Pcs0
Pcs[0]

enumerator kDSPI_Pcs1
Pcs[1]

enumerator kDSPI_Pcs2
Pcs[2]

enumerator kDSPI_Pcs3
Pcs[3]

enumerator kDSPI_Pcs4
Pcs[4]

enumerator kDSPI_Pcs5
Pcs[5]

enum _dspi_pcs_polarity_config
DSPI Peripheral Chip Select (Pcs) Polarity configuration.
Values:

enumerator kDSPI_PcsActiveHigh
Pcs Active High (idles low).

enumerator kDSPI_PcsActiveLow
Pcs Active Low (idles high).

enum _dspi_pcs_polarity
DSPI Peripheral Chip Select (Pcs) Polarity.
Values:

enumerator kDSPI_Pcs0ActiveLow
Pcs0 Active Low (idles high).

enumerator kDSPI_Pcs1ActiveLow
Pcs1 Active Low (idles high).

enumerator kDSPI_Pcs2ActiveLow
Pcs2 Active Low (idles high).

enumerator kDSPI_Pcs3ActiveLow

Pcs3 Active Low (idles high).

enumerator kDSPI_Pcs4ActiveLow

Pcs4 Active Low (idles high).

enumerator kDSPI_Pcs5ActiveLow

Pcs5 Active Low (idles high).

enumerator kDSPI_PcsAllActiveLow

Pcs0 to Pcs5 Active Low (idles high).

enum _dspi_clock_polarity

DSPI clock polarity configuration for a given CTAR.

Values:

enumerator kDSPI_ClockPolarityActiveHigh

CPOL=0. Active-high DSPI clock (idles low).

enumerator kDSPI_ClockPolarityActiveLow

CPOL=1. Active-low DSPI clock (idles high).

enum _dspi_clock_phase

DSPI clock phase configuration for a given CTAR.

Values:

enumerator kDSPI_ClockPhaseFirstEdge

CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

enumerator kDSPI_ClockPhaseSecondEdge

CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

enum _dspi_shift_direction

DSPI data shifter direction options for a given CTAR.

Values:

enumerator kDSPI_MsbFirst

Data transfers start with most significant bit.

enumerator kDSPI_LsbFirst

Data transfers start with least significant bit. Shifting out of LSB is not supported for slave

enum _dspi_delay_type

DSPI delay type selection.

Values:

enumerator kDSPI_PcsToSck

Pcs-to-SCK delay.

enumerator kDSPI_LastSckToPcs

The last SCK edge to Pcs delay.

enumerator kDSPI_BetweenTransfer

Delay between transfers.

enum _dspi_ctar_selection

DSPI Clock and Transfer Attributes Register (CTAR) selection.

Values:

enumerator kDSPI_Ctar0

CTAR0 selection option for master or slave mode; note that CTAR0 and CTAR0_SLAVE are the same register address.

enumerator kDSPI_Ctar1

CTAR1 selection option for master mode only.

enumerator kDSPI_Ctar2

CTAR2 selection option for master mode only; note that some devices do not support CTAR2.

enumerator kDSPI_Ctar3

CTAR3 selection option for master mode only; note that some devices do not support CTAR3.

enumerator kDSPI_Ctar4

CTAR4 selection option for master mode only; note that some devices do not support CTAR4.

enumerator kDSPI_Ctar5

CTAR5 selection option for master mode only; note that some devices do not support CTAR5.

enumerator kDSPI_Ctar6

CTAR6 selection option for master mode only; note that some devices do not support CTAR6.

enumerator kDSPI_Ctar7

CTAR7 selection option for master mode only; note that some devices do not support CTAR7.

enum _dspi_transfer_config_flag_for_master

Use this enumeration for the DSPI master transfer configFlags.

Values:

enumerator kDSPI_MasterCtar0
DSPI master CTAR shift macro; used internally. DSPI master transfer use CTAR0 setting.

enumerator kDSPI_MasterCtar1
DSPI master CTAR shift macro; used internally. DSPI master transfer use CTAR1 setting.

enumerator kDSPI_MasterCtar2
DSPI master CTAR shift macro; used internally. DSPI master transfer use CTAR2 setting.

enumerator kDSPI_MasterCtar3
DSPI master CTAR shift macro; used internally. DSPI master transfer use CTAR3 setting.

enumerator kDSPI_MasterCtar4
DSPI master CTAR shift macro; used internally. DSPI master transfer use CTAR4 setting.

enumerator kDSPI_MasterCtar5
DSPI master CTAR shift macro; used internally. DSPI master transfer use CTAR5 setting.

enumerator kDSPI_MasterCtar6
DSPI master CTAR shift macro; used internally. DSPI master transfer use CTAR6 setting.

enumerator kDSPI_MasterCtar7
DSPI master CTAR shift macro; used internally. DSPI master transfer use CTAR7 setting.

enumerator kDSPI_MasterPcs0
DSPI master PCS shift macro; used internally. DSPI master transfer use PCS0 signal.

enumerator kDSPI_MasterPcs1
DSPI master PCS shift macro; used internally. DSPI master transfer use PCS1 signal.

enumerator kDSPI_MasterPcs2
DSPI master PCS shift macro; used internally. DSPI master transfer use PCS2 signal.

enumerator kDSPI_MasterPcs3
DSPI master PCS shift macro; used internally. DSPI master transfer use PCS3 signal.

enumerator kDSPI_MasterPcs4
DSPI master PCS shift macro; used internally. DSPI master transfer use PCS4 signal.

enumerator kDSPI_MasterPcs5
DSPI master PCS shift macro; used internally. DSPI master transfer use PCS5 signal.

enumerator kDSPI_MasterPcsContinuous
Indicates whether the PCS signal is continuous.

enumerator kDSPI_MasterActiveAfterTransfer
Indicates whether the PCS signal is active after the last frame transfer.

enum _dspi_transfer_config_flag_for_slave

Use this enumeration for the DSPI slave transfer configFlags.

Values:

enumerator kDSPI_SlaveCtar0

DSPI slave CTAR shift macro; used internally. DSPI slave transfer use CTAR0 setting. DSPI slave can only use PCS0.

enum _dspi_transfer_state

DSPI transfer state, which is used for DSPI transactional API state machine.

Values:

enumerator kDSPI_Idle

Nothing in the transmitter/receiver.

enumerator kDSPI_Busy

Transfer queue is not finished.

enumerator kDSPI_Error

Transfer error.

typedef enum *_dspi_master_slave_mode* dspi_master_slave_mode_t

DSPI master or slave mode configuration.

typedef enum *_dspi_master_sample_point* dspi_master_sample_point_t

DSPI Sample Point: Controls when the DSPI master samples SIN in the Modified Transfer Format. This field is valid only when the CPHA bit in the CTAR register is 0.

typedef enum *_dspi_which_pcs_config* dspi_which_pcs_t

DSPI Peripheral Chip Select (Pcs) configuration (which Pcs to configure).

typedef enum *_dspi_pcs_polarity_config* dspi_pcs_polarity_config_t

DSPI Peripheral Chip Select (Pcs) Polarity configuration.

typedef enum *_dspi_clock_polarity* dspi_clock_polarity_t

DSPI clock polarity configuration for a given CTAR.

typedef enum *_dspi_clock_phase* dspi_clock_phase_t

DSPI clock phase configuration for a given CTAR.

typedef enum *_dspi_shift_direction* dspi_shift_direction_t

DSPI data shifter direction options for a given CTAR.

typedef enum *_dspi_delay_type* dspi_delay_type_t

DSPI delay type selection.

typedef enum *_dspi_ctar_selection* dspi_ctar_selection_t

DSPI Clock and Transfer Attributes Register (CTAR) selection.

typedef struct *_dspi_command_data_config* dspi_command_data_config_t

DSPI master command date configuration used for the SPIx_PUSHR.

typedef struct *_dspi_master_ctar_config* dspi_master_ctar_config_t

DSPI master ctar configuration structure.

typedef struct *_dspi_master_config* dspi_master_config_t

DSPI master configuration structure.

typedef struct *_dspi_slave_ctar_config* dspi_slave_ctar_config_t

DSPI slave ctar configuration structure.

typedef struct *_dspi_slave_config* dspi_slave_config_t

DSPI slave configuration structure.

typedef struct *_dspi_master_handle* dspi_master_handle_t

Forward declaration of the *_dspi_master_handle* typedefs.

The master handle.

typedef struct *_dspi_slave_handle* dspi_slave_handle_t

Forward declaration of the *_dspi_slave_handle* typedefs.

The slave handle.

typedef void (*dspi_master_transfer_callback_t)(SPI_Type *base, *dspi_master_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base
DSPI peripheral address.

Param handle
Pointer to the handle for the DSPI master.

Param status
Success or error code describing whether the transfer completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef void (*dspi_slave_transfer_callback_t)(SPI_Type *base, *dspi_slave_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base
DSPI peripheral address.

Param handle
Pointer to the handle for the DSPI slave.

Param status
Success or error code describing whether the transfer completed.

Param userData
Arbitrary pointer-dataSized value passed from the application.

typedef struct *_dspi_transfer* dspi_transfer_t

DSPI master/slave transfer structure.

typedef struct *_dspi_half_duplex_transfer* dspi_half_duplex_transfer_t

DSPI half-duplex(master) transfer structure.

volatile uint8_t g_dspiDummyData[]

Global variable for dummy data value setting.

DSPI_DUMMY_DATA

DSPI dummy data if there is no Tx data.

Dummy data used for Tx if there is no txData.

DSPI_MASTER_CTAR_SHIFT

DSPI master CTAR shift macro; used internally.

DSPI_MASTER_CTAR_MASK

DSPI master CTAR mask macro; used internally.

DSPI_MASTER_PCS_SHIFT

DSPI master PCS shift macro; used internally.

DSPI_MASTER_PCS_MASK

DSPI master PCS mask macro; used internally.

DSPI_SLAVE_CTAR_SHIFT

DSPI slave CTAR shift macro; used internally.

DSPI_SLAVE_CTAR_MASK

DSPI slave CTAR mask macro; used internally.

struct _dspi_command_data_config

#include <fsl_dspi.h> DSPI master command date configuration used for the SPIx_PUSHR.

Public Members

bool isPcsContinuous

Option to enable the continuous assertion of the chip select between transfers.

uint8_t whichCtar

The desired Clock and Transfer Attributes Register (CTAR) to use for CTAS.

uint8_t whichPcs

The desired PCS signal to use for the data transfer.

bool isEndOfQueue

Signals that the current transfer is the last in the queue.

bool clearTransferCount

Clears the SPI Transfer Counter (SPI_TCNT) before transmission starts.

struct _dspi_master_ctar_config

#include <fsl_dspi.h> DSPI master ctar configuration structure.

Public Members

uint32_t baudRate

Baud Rate for DSPI.

uint32_t bitsPerFrame

Bits per frame, minimum 4, maximum 16.

dspi_clock_polarity_t cpol

Clock polarity.

dspi_clock_phase_t cpha

Clock phase.

dspi_shift_direction_t direction

MSB or LSB data shift direction.

uint32_t pcsToSckDelayInNanoSec

PCS to SCK delay time in nanoseconds; setting to 0 sets the minimum delay. It also sets the boundary value if out of range.

uint32_t lastSckToPcsDelayInNanoSec

The last SCK to PCS delay time in nanoseconds; setting to 0 sets the minimum delay. It also sets the boundary value if out of range.

uint32_t betweenTransferDelayInNanoSec

After the SCK delay time in nanoseconds; setting to 0 sets the minimum delay. It also sets the boundary value if out of range.

struct _dspi_master_config

#include <fsl_dspi.h> DSPI master configuration structure.

Public Members

dspi_ctar_selection_t whichCtar

The desired CTAR to use.

dspi_master_ctar_config_t ctarConfig

Set the ctarConfig to the desired CTAR.

dspi_which_pcs_t whichPcs

The desired Peripheral Chip Select (pcs).

dspi_pcs_polarity_config_t pcsActiveHighOrLow

The desired PCS active high or low.

bool enableContinuousSCK

CONT_SCKE, continuous SCK enable. Note that the continuous SCK is only supported for CPHA = 1.

bool enableRxFifoOverWrite

ROOE, receive FIFO overflow overwrite enable. If ROOE = 0, the incoming data is ignored and the data from the transfer that generated the overflow is also ignored. If ROOE = 1, the incoming data is shifted to the shift register.

bool enableModifiedTimingFormat

Enables a modified transfer format to be used if true.

dspi_master_sample_point_t samplePoint

Controls when the module master samples SIN in the Modified Transfer Format. It's valid only when CPHA=0.

struct _dspi_slave_ctar_config

#include <fsl_dspi.h> DSPI slave ctar configuration structure.

Public Members

uint32_t bitsPerFrame

Bits per frame, minimum 4, maximum 16.

dspi_clock_polarity_t cpol

Clock polarity.

dspi_clock_phase_t cpha

Clock phase. Slave only supports MSB and does not support LSB.

struct _dspi_slave_config

#include <fsl_dspi.h> DSPI slave configuration structure.

Public Members

dspi_ctar_selection_t whichCtar

The desired CTAR to use.

dspi_slave_ctar_config_t ctarConfig

Set the ctarConfig to the desired CTAR.

bool enableContinuousSCK

CONT_SCKE, continuous SCK enable. Note that the continuous SCK is only supported for CPHA = 1.

bool enableRxFifoOverWrite

ROOE, receive FIFO overflow overwrite enable. If ROOE = 0, the incoming data is ignored and the data from the transfer that generated the overflow is also ignored. If ROOE = 1, the incoming data is shifted to the shift register.

bool enableModifiedTimingFormat

Enables a modified transfer format to be used if true.

dspi_master_sample_point_t samplePoint

Controls when the module master samples SIN in the Modified Transfer Format. It's valid only when CPHA=0.

struct __dspi__transfer

#include <fsl_dspi.h> DSPI master/slave transfer structure.

Public Members

const uint8_t *txData

Send buffer.

uint8_t *rxData

Receive buffer.

volatile size_t dataSize

Transfer bytes.

uint32_t configFlags

Transfer transfer configuration flags. Set from *_dspi_transfer_config_flag_for_master* if the transfer is used for master or *_dspi_transfer_config_flag_for_slave* enumeration if the transfer is used for slave.

struct __dspi__half__duplex__transfer

#include <fsl_dspi.h> DSPI half-duplex(master) 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; set from *_dspi_transfer_config_flag_for_master*.

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 __dspi_master_handle
: *#include <fsl_dspi.h>* DSPI master transfer handle structure used for transactional API.

Public Members

uint32_t bitsPerFrame
: The desired number of bits per frame.

volatile uint32_t command
: The desired data command.

volatile uint32_t lastCommand
: The desired last data command.

uint8_t fifoSize
: FIFO dataSize.

volatile bool isPcsActiveAfterTransfer
: Indicates whether the PCS signal is active after the last frame transfer.

volatile bool isThereExtraByte
: Indicates whether there are extra bytes.

const uint8_t *volatile txData
: Send buffer.

uint8_t *volatile rxData
: Receive buffer.

volatile size_t remainingSendByteCount
: A number of bytes remaining to send.

volatile size_t remainingReceiveByteCount
A number of bytes remaining to receive.

size_t totalByteCount
A number of transfer bytes

volatile uint8_t state
DSPI transfer state, see *_dspi_transfer_state*.

dspi_master_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

struct __dspi_slave_handle
#include <fsl_dspi.h> DSPI slave transfer handle structure used for the transactional API.

Public Members

uint32_t bitsPerFrame
The desired number of bits per frame.

volatile bool isThereExtraByte
Indicates whether there are extra bytes.

const uint8_t *volatile txData
Send buffer.

uint8_t *volatile rxData
Receive buffer.

volatile size_t remainingSendByteCount
A number of bytes remaining to send.

volatile size_t remainingReceiveByteCount
A number of bytes remaining to receive.

size_t totalByteCount
A number of transfer bytes

volatile uint8_t state
DSPI transfer state.

volatile uint32_t errorCount
Error count for slave transfer.

dspi_slave_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

2.9 DSPI eDMA Driver

void DSPI_MasterTransferCreateHandleEDMA(SPI_Type *base, *dspi_master_edma_handle_t* *handle, *dspi_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToIntermediaryHandle, *edma_handle_t* *edmaIntermediaryToTxRegHandle)

Initializes the DSPI master eDMA handle.

This function initializes the DSPI eDMA handle which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Note: DSPI eDMA has separated (RX and TX as two sources) or shared (RX and TX are the same source) DMA request source.

- For the separated DMA request source, enable and set the RX DMAMUX source for edmaRxRegToRxDataHandle and TX DMAMUX source for edmaIntermediaryToTxRegHandle.
- For the shared DMA request source, enable and set the RX/RX DMAMUX source for the edmaRxRegToRxDataHandle.

Parameters

- base – DSPI peripheral base address.
- handle – DSPI handle pointer to *_dspi_master_edma_handle*.
- callback – DSPI callback.
- userData – A callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToIntermediaryHandle – edmaTxDataToIntermediaryHandle pointer to edma_handle_t.
- edmaIntermediaryToTxRegHandle – edmaIntermediaryToTxRegHandle pointer to edma_handle_t.

status_t DSPI_MasterTransferEDMA(SPI_Type *base, *dspi_master_edma_handle_t* *handle, *dspi_transfer_t* *transfer)

DSPI master transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The max transfer size of each transfer depends on whether the instance's Tx/Rx shares the same DMA request. If **FSL_FEATURE_DSPI_HAS_SEPARATE_DMA_RX_TX_REQn(x)** is true, then the max transfer size is 32767 datawidth of data, otherwise is 511.

Parameters

- base – DSPI peripheral base address.
- handle – A pointer to the *_dspi_master_edma_handle* structure which stores the transfer state.
- transfer – A pointer to the *dspi_transfer_t* structure.

Returns
status of status_t.

status_t DSPI_MasterHalfDuplexTransferEDMA(SPI_Type *base, *dspi_master_edma_handle_t* *handle, *dspi_half_duplex_transfer_t* *xfer)

Transfers a block of data using a eDMA method.

This function transfers data using eDNA, the transfer mechanism is half-duplex. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

- base – DSPI base pointer
- handle – A pointer to the *_dspi_master_edma_handle* structure which stores the transfer state.
- xfer – A pointer to the *dspi_half_duplex_transfer_t* structure.

Returns
status of status_t.

void DSPI_MasterTransferAbortEDMA(SPI_Type *base, *dspi_master_edma_handle_t* *handle)

DSPI master aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – DSPI peripheral base address.
- handle – A pointer to the *_dspi_master_edma_handle* structure which stores the transfer state.

status_t DSPI_MasterTransferGetCountEDMA(SPI_Type *base, *dspi_master_edma_handle_t* *handle, size_t *count)

Gets the master eDMA transfer count.

This function gets the master eDMA transfer count.

Parameters

- base – DSPI peripheral base address.
- handle – A pointer to the *_dspi_master_edma_handle* structure which stores the transfer state.
- count – A number of bytes transferred by the non-blocking transaction.

Returns
status of status_t.

void DSPI_SlaveTransferCreateHandleEDMA(SPI_Type *base, *dspi_slave_edma_handle_t* *handle, *dspi_slave_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaRxRegToRxDataHandle, *edma_handle_t* *edmaTxDataToTxRegHandle)

Initializes the DSPI slave eDMA handle.

This function initializes the DSPI eDMA handle which can be used for other DSPI transactional APIs. Usually, for a specified DSPI instance, call this API once to get the initialized handle.

Note: DSPI eDMA has separated (RN and TX in 2 sources) or shared (RX and TX are the same source) DMA request source.

- For the separated DMA request source, enable and set the RX DMAMUX source for edmaRxRegToRxDataHandle and TX DMAMUX source for edmaTxDataToTxRegHandle.
- For the shared DMA request source, enable and set the RX/RX DMAMUX source for the edmaRxRegToRxDataHandle.

Parameters

- base – DSPI peripheral base address.
- handle – DSPI handle pointer to *_dspi_slave_edma_handle*.
- callback – DSPI callback.
- userData – A callback function parameter.
- edmaRxRegToRxDataHandle – edmaRxRegToRxDataHandle pointer to edma_handle_t.
- edmaTxDataToTxRegHandle – edmaTxDataToTxRegHandle pointer to edma_handle_t.

status_t DSPI_SlaveTransferEDMA(SPI_Type *base, *dspi_slave_edma_handle_t* *handle, *dspi_transfer_t* *transfer)

DSPI slave transfer data using eDMA.

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called. Note that the slave eDMA transfer doesn't support transfer_size is 1 when the bitsPerFrame is greater than eight.

Note: The max transfer size of each transfer depends on whether the instance's Tx/Rx shares the same DMA request. If **FSL_FEATURE_DSPI_HAS_SEPARATE_DMA_RX_TX_REQn(x)** is true, then the max transfer size is 32767 datawidth of data, otherwise is 511.

Parameters

- base – DSPI peripheral base address.
- handle – A pointer to the *_dspi_slave_edma_handle* structure which stores the transfer state.
- transfer – A pointer to the *dspi_transfer_t* structure.

Returns

status of status_t.

void DSPI_SlaveTransferAbortEDMA(SPI_Type *base, *dspi_slave_edma_handle_t* *handle)

DSPI slave aborts a transfer which is using eDMA.

This function aborts a transfer which is using eDMA.

Parameters

- base – DSPI peripheral base address.
- handle – A pointer to the *_dspi_slave_edma_handle* structure which stores the transfer state.

status_t DSPI_SlaveTransferGetCountEDMA(SPI_Type *base, *dspi_slave_edma_handle_t* *handle, size_t *count)

Gets the slave eDMA transfer count.

This function gets the slave eDMA transfer count.

Parameters

- base – DSPI peripheral base address.
- handle – A pointer to the *_dspi_slave_edma_handle* structure which stores the transfer state.
- count – A number of bytes transferred so far by the non-blocking transaction.

Returns

status of status_t.

FSL_DSPI_EDMA_DRIVER_VERSION

DSPI EDMA driver version 2.2.7.

typedef struct *_dspi_master_edma_handle* dspi_master_edma_handle_t

Forward declaration of the DSPI eDMA master handle typedefs.

typedef struct *_dspi_slave_edma_handle* dspi_slave_edma_handle_t

Forward declaration of the DSPI eDMA slave handle typedefs.

typedef void (*dspi_master_edma_transfer_callback_t)(SPI_Type *base, *dspi_master_edma_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base
DSPI peripheral base address.

Param handle
A pointer to the handle for the DSPI master.

Param status
Success or error code describing whether the transfer completed.

Param userData
An arbitrary pointer-dataSized value passed from the application.

typedef void (*dspi_slave_edma_transfer_callback_t)(SPI_Type *base, *dspi_slave_edma_handle_t* *handle, *status_t* status, void *userData)

Completion callback function pointer type.

Param base
DSPI peripheral base address.

Param handle
A pointer to the handle for the DSPI slave.

Param status
Success or error code describing whether the transfer completed.

Param userData
An arbitrary pointer-dataSized value passed from the application.

DSPI_EDMA_MAX_TRANSFER_SIZE(base, width)

DSPI EDMA max transfer data size calculate.

Parameters

- base – DSPI peripheral base address.
- width – Transfer width

struct _dspi_master_edma_handle

#include <fsl_dspi_edma.h> DSPI master eDMA transfer handle structure used for the transactional API.

Public Members

uint32_t bitsPerFrame

The desired number of bits per frame.

volatile uint32_t command

The desired data command.

volatile uint32_t lastCommand

The desired last data command.

uint8_t fifoSize

FIFO dataSize.

volatile bool isPcsActiveAfterTransfer

Indicates whether the PCS signal keeps active after the last frame transfer.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

volatile uint8_t state

DSPI transfer state, see *_dspi_transfer_state*.

const uint8_t *volatile txData

Send buffer.

uint8_t *volatile rxData

Receive buffer.

volatile size_t remainingSendByteCount

A number of bytes remaining to send.

volatile size_t remainingReceiveByteCount
A number of bytes remaining to receive.

size_t totalByteCount
A number of transfer bytes

uint32_t rxBuffIfNull
Used if there is not rxData for DMA purpose.

uint32_t txBuffIfNull
Used if there is not txData for DMA purpose.

dspi_master_edma_transfer_callback_t callback
Completion callback.

void *userData
Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToIntermediaryHandle
edma_handle_t handle point used for TxData to Intermediary

edma_handle_t *edmaIntermediaryToTxRegHandle
edma_handle_t handle point used for Intermediary to TxReg

edma_tcd_t dspiSoftwareTCD[2]
SoftwareTCD , internal used

struct __dspi_slave_edma_handle
#include <fsl_dspi_edma.h> DSPI slave eDMA transfer handle structure used for the transactional API.

Public Members

uint32_t bitsPerFrame
The desired number of bits per frame.

const uint8_t *volatile txData
Send buffer.

uint8_t *volatile rxData
Receive buffer.

volatile size_t remainingSendByteCount
A number of bytes remaining to send.

volatile size_t remainingReceiveByteCount
: A number of bytes remaining to receive.

size_t totalByteCount
: A number of transfer bytes

uint32_t rxBuffIfNull
: Used if there is not rxData for DMA purpose.

uint32_t txBuffIfNull
: Used if there is not txData for DMA purpose.

uint32_t txLastData
: Used if there is an extra byte when 16bits per frame for DMA purpose.

uint8_t nbytes
: eDMA minor byte transfer count initially configured.

volatile uint8_t state
: DSPI transfer state.

dspi_slave_edma_transfer_callback_t callback
: Completion callback.

void *userData
: Callback user data.

edma_handle_t *edmaRxRegToRxDataHandle
: edma_handle_t handle point used for RxReg to RxData buff

edma_handle_t *edmaTxDataToTxRegHandle
: edma_handle_t handle point used for TxData to TxReg

2.10 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

void EDMA_Init(DMA_Type *base, const *edma_config_t* *config)
: Initializes the eDMA peripheral.

  This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure. All emda enabled request will be cleared in this function.

  Note: This function enables the minor loop map feature.

  Parameters

  - base – eDMA peripheral base address.
  - config – A pointer to the configuration structure, see “edma_config_t”.

void EDMA_Deinit(DMA_Type *base)

Deinitializes the eDMA peripheral.

This function gates the eDMA clock.

Parameters

- base – eDMA peripheral base address.

void EDMA_InstallTCD(DMA_Type *base, uint32_t channel, *edma_tcd_t* *tcd)

Push content of TCD structure into hardware TCD register.

Parameters

- base – EDMA peripheral base address.
- channel – EDMA channel number.
- tcd – Point to TCD structure.

void EDMA_GetDefaultConfig(*edma_config_t* *config)

Gets the eDMA default configuration structure.

This function sets the configuration structure to default values. The default configuration is set to the following values.

```
config.enableContinuousLinkMode = false;
config.enableHaltOnError = true;
config.enableRoundRobinArbitration = false;
config.enableDebugMode = false;
```

Parameters

- config – A pointer to the eDMA configuration structure.

static inline void EDMA_EnableContinuousChannelLinkMode(DMA_Type *base, bool enable)

Enable/Disable continuous channel link mode.

Note: Do not use continuous link mode with a channel linking to itself if there is only one minor loop iteration per service request, for example, if the channel's NBYTES value is the same as either the source or destination size. The same data transfer profile can be achieved by simply increasing the NBYTES value, which provides more efficient, faster processing.

Parameters

- base – EDMA peripheral base address.
- enable – true is enable, false is disable.

static inline void EDMA_EnableMinorLoopMapping(DMA_Type *base, bool enable)

Enable/Disable minor loop mapping.

The TCDn.word2 is redefined to include individual enable fields, an offset field, and the NBYTES field.

Parameters

- base – EDMA peripheral base address.
- enable – true is enable, false is disable.

void EDMA_ResetChannel(DMA_Type *base, uint32_t channel)

Sets all TCD registers to default values.

This function sets TCD registers for this channel to default values.

Note: This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

Note: This function enables the auto stop request feature.

Parameters

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

void EDMA_SetTransferConfig(DMA_Type *base, uint32_t channel, const *edma_transfer_config_t* *config, *edma_tcd_t* *nextTcd)

Configures the eDMA transfer attribute.

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address. Example:

```
edma_transfer_t config;
edma_tcd_t tcd;
config.srcAddr = ..;
config.destAddr = ..;
...
EDMA_SetTransferConfig(DMA0, channel, &config, &stcd);
```

Note: If nextTcd is not NULL, it means scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the eDMA_ResetChannel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Point to TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

void EDMA_SetMinorOffsetConfig(DMA_Type *base, uint32_t channel, const *edma_minor_offset_config_t* *config)

Configures the eDMA minor offset feature.

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- config – A pointer to the minor offset configuration structure.

void EDMA_SetChannelPreemptionConfig(DMA_Type *base, uint32_t channel, const *edma_channel_Preemption_config_t* *config)

Configures the eDMA channel preemption feature.

This function configures the channel preemption attribute and the priority of the channel.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number
- config – A pointer to the channel preemption configuration structure.

void EDMA_SetChannelLink(DMA_Type *base, uint32_t channel, *edma_channel_link_type_t* linkType, uint32_t linkedChannel)

Sets the channel link for the eDMA transfer.

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- linkType – A channel link type, which can be one of the following:
  - kEDMA_LinkNone
  - kEDMA_MinorLink
  - kEDMA_MajorLink
- linkedChannel – The linked channel number.

void EDMA_SetBandWidth(DMA_Type *base, uint32_t channel, *edma_bandwidth_t* bandWidth)

Sets the bandwidth for the eDMA transfer.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- bandWidth – A bandwidth setting, which can be one of the following:
  - kEDMABandwidthStallNone
  - kEDMABandwidthStall4Cycle
  - kEDMABandwidthStall8Cycle

void EDMA_SetModulo(DMA_Type *base, uint32_t channel, *edma_modulo_t* srcModulo, *edma_modulo_t* destModulo)

Sets the source modulo and the destination modulo for the eDMA transfer.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

static inline void EDMA_EnableAsyncRequest(DMA_Type *base, uint32_t channel, bool enable)

Enables an async request for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

static inline void EDMA_EnableAutoStopRequest(DMA_Type *base, uint32_t channel, bool enable)

Enables an auto stop request for the eDMA transfer.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- enable – The command to enable (true) or disable (false).

void EDMA_EnableChannelInterrupts(DMA_Type *base, uint32_t channel, uint32_t mask)

Enables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

void EDMA_DisableChannelInterrupts(DMA_Type *base, uint32_t channel, uint32_t mask)

Disables the interrupt source for the eDMA transfer.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of the interrupt source to be set. Use the defined edma_interrupt_enable_t type.

void EDMA_SetMajorOffsetConfig(DMA_Type *base, uint32_t channel, int32_t sourceOffset, int32_t destOffset)

Configures the eDMA channel TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- base – eDMA peripheral base address.
- channel – edma channel number.
- sourceOffset – source address offset will be applied to source address after major loop done.
- destOffset – destination address offset will be applied to source address after major loop done.

void EDMA_TcdReset(*edma_tcd_t* *tcd)

Sets all fields to default values for the TCD structure.

This function sets all fields for this TCD structure to default value.

Note: This function enables the auto stop request feature.

Parameters

- tcd – Pointer to the TCD structure.

void EDMA_TcdSetTransferConfig(*edma_tcd_t* *tcd, const *edma_transfer_config_t* *config, *edma_tcd_t* *nextTcd)

Configures the eDMA TCD transfer attribute.

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The TCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
edma_transfer_t config = {
...
}
edma_tcd_t tcd __aligned(32);
edma_tcd_t nextTcd __aligned(32);
EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
```

Note: TCD address should be 32 bytes aligned or it causes an eDMA error.

Note: If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA_TcdReset.

Parameters

- tcd – Pointer to the TCD structure.
- config – Pointer to eDMA transfer configuration structure.
- nextTcd – Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature.

void EDMA_TcdSetMinorOffsetConfig(*edma_tcd_t* *tcd, const *edma_minor_offset_config_t* *config)

Configures the eDMA TCD minor offset feature.

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

Parameters

- tcd – A point to the TCD structure.
- config – A pointer to the minor offset configuration structure.

void EDMA_TcdSetChannelLink(*edma_tcd_t* *tcd, *edma_channel_link_type_t* linkType, uint32_t linkedChannel)

Sets the channel link for the eDMA TCD.

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note: Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

- tcd – Point to the TCD structure.
- linkType – Channel link type, it can be one of:
  - kEDMA_LinkNone
  - kEDMA_MinorLink
  - kEDMA_MajorLink
- linkedChannel – The linked channel number.

static inline void EDMA_TcdSetBandWidth(*edma_tcd_t* *tcd, *edma_bandwidth_t* bandWidth)

Sets the bandwidth for the eDMA TCD.

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

- tcd – A pointer to the TCD structure.
- bandWidth – A bandwidth setting, which can be one of the following:
  - kEDMABandwidthStallNone
  - kEDMABandwidthStall4Cycle
  - kEDMABandwidthStall8Cycle

void EDMA_TcdSetModulo(*edma_tcd_t* *tcd, *edma_modulo_t* srcModulo, *edma_modulo_t* destModulo)

Sets the source modulo and the destination modulo for the eDMA TCD.

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

- tcd – A pointer to the TCD structure.
- srcModulo – A source modulo value.
- destModulo – A destination modulo value.

static inline void EDMA_TcdEnableAutoStopRequest(*edma_tcd_t* *tcd, bool enable)

Sets the auto stop request for the eDMA TCD.

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

- tcd – A pointer to the TCD structure.
- enable – The command to enable (true) or disable (false).

void EDMA_TcdEnableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Enables the interrupt source for the eDMA TCD.

Parameters

- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

void EDMA_TcdDisableInterrupts(*edma_tcd_t* *tcd, uint32_t mask)

Disables the interrupt source for the eDMA TCD.

Parameters

- tcd – Point to the TCD structure.
- mask – The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type.

void EDMA_TcdSetMajorOffsetConfig(*edma_tcd_t* *tcd, int32_t sourceOffset, int32_t destOffset)

Configures the eDMA TCD major offset feature.

Adjustment value added to the source address at the completion of the major iteration count

Parameters

- tcd – A point to the TCD structure.
- sourceOffset – source address offset wiil be applied to source address after major loop done.
- destOffset – destination address offset will be applied to source address after major loop done.

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

Enables the eDMA hardware channel request.

This function enables the hardware channel request.

Parameters

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

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

Disables the eDMA hardware channel request.

This function disables the hardware channel request.

Parameters

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

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

Starts the eDMA transfer by using the software trigger.

This function starts a minor loop transfer.

Parameters

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

uint32_t EDMA_GetRemainingMajorLoopCount(DMA_Type *base, uint32_t channel)

Gets the remaining major loop count from the eDMA current channel TCD.

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

Note: 1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.

a. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA_TCDn_NBYTES_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: RemainingBytes = RemainingMajorLoopCount * NBYTES(initially configured)

Parameters

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

Returns

Major loop count which has not been transferred yet for the current TCD.

static inline uint32_t EDMA_GetErrorStatusFlags(DMA_Type *base)

Gets the eDMA channel error status flags.

Parameters

- base – eDMA peripheral base address.

Returns

The mask of error status flags. Users need to use the _edma_error_status_flags type to decode the return variables.

uint32_t EDMA_GetChannelStatusFlags(DMA_Type *base, uint32_t channel)

Gets the eDMA channel status flags.

Parameters

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

Returns

The mask of channel status flags. Users need to use the _edma_channel_status_flags type to decode the return variables.

void EDMA_ClearChannelStatusFlags(DMA_Type *base, uint32_t channel, uint32_t mask)

Clears the eDMA channel status flags.

Parameters

- base – eDMA peripheral base address.
- channel – eDMA channel number.
- mask – The mask of channel status to be cleared. Users need to use the defined _edma_channel_status_flags type.

void EDMA_CreateHandle(*edma_handle_t* *handle, DMA_Type *base, uint32_t channel)

Creates the eDMA handle.

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

- handle – eDMA handle pointer. The eDMA handle stores callback function and parameters.
- base – eDMA peripheral base address.
- channel – eDMA channel number.

void EDMA_InstallTCDMemory(*edma_handle_t* *handle, *edma_tcd_t* *tcdPool, uint32_t tcdSize)

Installs the TCDs memory pool into the eDMA handle.

This function is called after the EDMA_CreateHandle to use scatter/gather feature. This function shall only be used while users need to use scatter gather mode. Scatter gather mode enables EDMA to load a new transfer control block (tcd) in hardware, and automatically reconfigure that DMA channel for a new transfer. Users need to prepare tcd memory and also configure tcds using interface EDMA_SubmitTransfer.

Parameters

- handle – eDMA handle pointer.
- tcdPool – A memory pool to store TCDs. It must be 32 bytes aligned.
- tcdSize – The number of TCD slots.

void EDMA_SetCallback(*edma_handle_t* *handle, *edma_callback* callback, void *userData)

Installs a callback function for the eDMA transfer.

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes. This function will be called every time one tcd finished transfer.

Parameters

- handle – eDMA handle pointer.
- callback – eDMA callback function pointer.
- userData – A parameter for the callback function.

void EDMA_PrepareTransferConfig(*edma_transfer_config_t* *config, void *srcAddr, uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes)

Prepares the eDMA transfer structure configurations.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- config – The user configuration structure of type edma_transfer_t.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- srcOffset – source address offset.
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- destOffset – destination address offset.
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.

void EDMA_PrepareTransfer(*edma_transfer_config_t* *config, void *srcAddr, uint32_t srcWidth, void *destAddr, uint32_t destWidth, uint32_t bytesEachRequest, uint32_t transferBytes, *edma_transfer_type_t* transferType)

Prepares the eDMA transfer structure.

This function prepares the transfer configuration structure according to the user input.

Note: The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

Parameters

- config – The user configuration structure of type edma_transfer_t.
- srcAddr – eDMA transfer source address.
- srcWidth – eDMA transfer source address width(bytes).
- destAddr – eDMA transfer destination address.
- destWidth – eDMA transfer destination address width(bytes).
- bytesEachRequest – eDMA transfer bytes per channel request.
- transferBytes – eDMA transfer bytes to be transferred.
- transferType – eDMA transfer type.

status_t EDMA_SubmitTransfer(*edma_handle_t* *handle, const *edma_transfer_config_t* *config)

Submits the eDMA transfer request.

This function submits the eDMA transfer request according to the transfer configuration structure. In scatter gather mode, call this function will add a configured tcd to the circular list of tcd pool. The tcd pools is setup by call function EDMA_InstallTCDMemory before.

Parameters

- handle – eDMA handle pointer.
- config – Pointer to eDMA transfer configuration structure.

Return values

- kStatus_EDMA_Success – It means submit transfer request succeed.
- kStatus_EDMA_QueueFull – It means TCD queue is full. Submit transfer request is not allowed.
- kStatus_EDMA_Busy – It means the given channel is busy, need to submit request later.

void EDMA_StartTransfer(*edma_handle_t* *handle)

eDMA starts transfer.

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

Parameters

- handle – eDMA handle pointer.

void EDMA_StopTransfer(*edma_handle_t* *handle)

eDMA stops transfer.

This function disables the channel request to pause the transfer. Users can call *EDMA_StartTransfer()* again to resume the transfer.

Parameters

- handle – eDMA handle pointer.

void EDMA_AbortTransfer(*edma_handle_t* *handle)

eDMA aborts transfer.

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

- handle – DMA handle pointer.

static inline uint32_t EDMA_GetUnusedTCDNumber(*edma_handle_t* *handle)

Get unused TCD slot number.

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

- handle – DMA handle pointer.

Returns

The unused tcd slot number.

static inline uint32_t EDMA_GetNextTCDAddress(*edma_handle_t* *handle)

Get the next tcd address.

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

- handle – DMA handle pointer.

Returns

The next TCD address.

void EDMA_HandleIRQ(*edma_handle_t* *handle)

eDMA IRQ handler for the current major loop transfer completion.

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Note: For the case using TCD queue, when the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor (if scatter/gather is enabled).

For instance, when the time interrupt of TCD[0] happens, the TCD[1] has already been loaded into the eDMA engine. As sga and sga_index are calculated based on the DLAST_SGA bitfield lies in the TCD_CSR register, the sga_index in this case should be 2 (DLAST_SGA of TCD[1] stores the address of TCD[2]). Thus, the "tcdUsed" updated should be (tcdUsed - 2U) which indicates the number of TCDs can be loaded in the memory pool (because TCD[0] and TCD[1] have been loaded into the eDMA engine at this point already.).

For the last two continuous ISRs in a scatter/gather process, they both load the last TCD (The last ISR does not load a new TCD) from the memory pool to the eDMA engine when major loop completes. Therefore, ensure that the header and tcdUsed updated are identical for them. tcdUsed are both 0 in this case as no TCD to be loaded.

See the "eDMA basic data flow" in the eDMA Functional description section of the Reference Manual for further details.

Parameters

- handle – eDMA handle pointer.

void EDMA_DriverIRQHandler(uint32_t instance, uint32_t channel)

EDMA IRQ handler for a given instance and channel.

This is a common parameterized IRQ handler entry that dispatches to the appropriate channel handle based on the instance and channel number.

Parameters

- instance – EDMA instance number.
- channel – EDMA channel number.

FSL_EDMA_DRIVER_VERSION

eDMA driver version

Version 2.4.9.

enum _edma_transfer_size

eDMA transfer configuration

Values:

enumerator kEDMA_TransferSize1Bytes

Source/Destination data transfer size is 1 byte every time

enumerator kEDMA_TransferSize2Bytes

Source/Destination data transfer size is 2 bytes every time

enumerator kEDMA_TransferSize4Bytes

Source/Destination data transfer size is 4 bytes every time

enumerator kEDMA_TransferSize8Bytes

Source/Destination data transfer size is 8 bytes every time

enumerator kEDMA_TransferSize16Bytes
Source/Destination data transfer size is 16 bytes every time

enumerator kEDMA_TransferSize32Bytes
Source/Destination data transfer size is 32 bytes every time

enum _edma_modulo
eDMA modulo configuration

Values:

enumerator kEDMA_ModuloDisable
Disable modulo

enumerator kEDMA_Modulo2bytes
Circular buffer size is 2 bytes.

enumerator kEDMA_Modulo4bytes
Circular buffer size is 4 bytes.

enumerator kEDMA_Modulo8bytes
Circular buffer size is 8 bytes.

enumerator kEDMA_Modulo16bytes
Circular buffer size is 16 bytes.

enumerator kEDMA_Modulo32bytes
Circular buffer size is 32 bytes.

enumerator kEDMA_Modulo64bytes
Circular buffer size is 64 bytes.

enumerator kEDMA_Modulo128bytes
Circular buffer size is 128 bytes.

enumerator kEDMA_Modulo256bytes
Circular buffer size is 256 bytes.

enumerator kEDMA_Modulo512bytes
Circular buffer size is 512 bytes.

enumerator kEDMA_Modulo1Kbytes
Circular buffer size is 1 K bytes.

enumerator kEDMA_Modulo2Kbytes
Circular buffer size is 2 K bytes.

enumerator kEDMA_Modulo4Kbytes
Circular buffer size is 4 K bytes.

enumerator kEDMA_Modulo8Kbytes
Circular buffer size is 8 K bytes.

enumerator kEDMA_Modulo16Kbytes
Circular buffer size is 16 K bytes.

enumerator kEDMA_Modulo32Kbytes
Circular buffer size is 32 K bytes.

enumerator kEDMA_Modulo64Kbytes
Circular buffer size is 64 K bytes.

enumerator kEDMA_Modulo128Kbytes
Circular buffer size is 128 K bytes.

enumerator kEDMA_Modulo256Kbytes
Circular buffer size is 256 K bytes.

enumerator kEDMA_Modulo512Kbytes
Circular buffer size is 512 K bytes.

enumerator kEDMA_Modulo1Mbytes
Circular buffer size is 1 M bytes.

enumerator kEDMA_Modulo2Mbytes
Circular buffer size is 2 M bytes.

enumerator kEDMA_Modulo4Mbytes
Circular buffer size is 4 M bytes.

enumerator kEDMA_Modulo8Mbytes
Circular buffer size is 8 M bytes.

enumerator kEDMA_Modulo16Mbytes
Circular buffer size is 16 M bytes.

enumerator kEDMA_Modulo32Mbytes
Circular buffer size is 32 M bytes.

enumerator kEDMA_Modulo64Mbytes
Circular buffer size is 64 M bytes.

enumerator kEDMA_Modulo128Mbytes
Circular buffer size is 128 M bytes.

enumerator kEDMA_Modulo256Mbytes
Circular buffer size is 256 M bytes.

enumerator kEDMA_Modulo512Mbytes
Circular buffer size is 512 M bytes.

enumerator kEDMA_Modulo1Gbytes
Circular buffer size is 1 G bytes.

enumerator kEDMA_Modulo2Gbytes
Circular buffer size is 2 G bytes.

enum _edma_bandwidth
Bandwidth control.
Values:

enumerator kEDMA_BandwidthStallNone
No eDMA engine stalls.

enumerator kEDMA_BandwidthStall4Cycle
eDMA engine stalls for 4 cycles after each read/write.

enumerator kEDMA_BandwidthStall8Cycle
eDMA engine stalls for 8 cycles after each read/write.

enum _edma_channel_link_type
Channel link type.
Values:

enumerator kEDMA_LinkNone
No channel link

enumerator kEDMA_MinorLink
Channel link after each minor loop

enumerator kEDMA_MajorLink
Channel link while major loop count exhausted

_edma_channel_status_flags eDMA channel status flags.
Values:

enumerator kEDMA_DoneFlag
DONE flag, set while transfer finished, CITER value exhausted

enumerator kEDMA_ErrorFlag
eDMA error flag, an error occurred in a transfer

enumerator kEDMA_InterruptFlag
eDMA interrupt flag, set while an interrupt occurred of this channel

_edma_error_status_flags eDMA channel error status flags.

Values:

enumerator kEDMA_DestinationBusErrorFlag
Bus error on destination address

enumerator kEDMA_SourceBusErrorFlag
Bus error on the source address

enumerator kEDMA_ScatterGatherErrorFlag
Error on the Scatter/Gather address, not 32byte aligned.

enumerator kEDMA_NbytesErrorFlag
NBYTES/CITER configuration error

enumerator kEDMA_DestinationOffsetErrorFlag
Destination offset not aligned with destination size

enumerator kEDMA_DestinationAddressErrorFlag
Destination address not aligned with destination size

enumerator kEDMA_SourceOffsetErrorFlag
Source offset not aligned with source size

enumerator kEDMA_SourceAddressErrorFlag
Source address not aligned with source size

enumerator kEDMA_ErrorChannelFlag
Error channel number of the cancelled channel number

enumerator kEDMA_ChannelPriorityErrorFlag
Channel priority is not unique.

enumerator kEDMA_TransferCanceledFlag
Transfer cancelled

enumerator kEDMA_ValidFlag
No error occurred, this bit is 0. Otherwise, it is 1.

enum _edma_interrupt_enable
eDMA interrupt source

Values:

enumerator kEDMA_ErrorInterruptEnable
Enable interrupt while channel error occurs.

enumerator kEDMA_MajorInterruptEnable
Enable interrupt while major count exhausted.

enumerator kEDMA_HalfInterruptEnable
Enable interrupt while major count to half value.

enum _edma_transfer_type
eDMA transfer type
Values:

enumerator kEDMA_MemoryToMemory
Transfer from memory to memory

enumerator kEDMA_PeripheralToMemory
Transfer from peripheral to memory

enumerator kEDMA_MemoryToPeripheral
Transfer from memory to peripheral

enumerator kEDMA_PeripheralToPeripheral
Transfer from Peripheral to peripheral

_edma_transfer_status eDMA transfer status
Values:

enumerator kStatus_EDMA_QueueFull
TCD queue is full.

enumerator kStatus_EDMA_Busy
Channel is busy and can't handle the transfer request.

typedef enum *_edma_transfer_size* edma_transfer_size_t
eDMA transfer configuration

typedef enum *_edma_modulo* edma_modulo_t
eDMA modulo configuration

typedef enum *_edma_bandwidth* edma_bandwidth_t
Bandwidth control.

typedef enum *_edma_channel_link_type* edma_channel_link_type_t
Channel link type.

typedef enum *_edma_interrupt_enable* edma_interrupt_enable_t
eDMA interrupt source

typedef enum *_edma_transfer_type* edma_transfer_type_t

eDMA transfer type

typedef struct *_edma_config* edma_config_t

eDMA global configuration structure.

typedef struct *_edma_transfer_config* edma_transfer_config_t

eDMA transfer configuration

This structure configures the source/destination transfer attribute.

typedef struct *_edma_channel_Preemption_config* edma_channel_Preemption_config_t

eDMA channel priority configuration

typedef struct *_edma_minor_offset_config* edma_minor_offset_config_t

eDMA minor offset configuration

typedef struct *_edma_tcd* edma_tcd_t

eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

typedef void (*edma_callback)(struct *_edma_handle* *handle, void *userData, bool transferDone, uint32_t tcds)

Define callback function for eDMA.

This callback function is called in the EDMA interrupt handle. In normal mode, run into callback function means the transfer users need is done. In scatter gather mode, run into callback function means a transfer control block (tcd) is finished. Not all transfer finished, users can get the finished tcd numbers using interface EDMA_GetUnusedTCDNumber.

Param handle
EDMA handle pointer, users shall not touch the values inside.

Param userData
The callback user parameter pointer. Users can use this parameter to involve things users need to change in EDMA callback function.

Param transferDone
If the current loaded transfer done. In normal mode it means if all transfer done. In scatter gather mode, this parameter shows is the current transfer block in EDMA register is done. As the load of core is different, it will be different if the new tcd loaded into EDMA registers while this callback called. If true, it always means new tcd still not loaded into registers, while false means new tcd already loaded into registers.

Param tcds
How many tcds are done from the last callback. This parameter only used in scatter gather mode. It tells user how many tcds are finished between the last callback and this.

typedef struct *_edma_handle* edma_handle_t

eDMA transfer handle structure

DMA_DCHPRI_INDEX(channel)

Compute the offset unit from DCHPRI3.

struct __edma__config

#include <fsl_edma.h> eDMA global configuration structure.

Public Members

bool enableContinuousLinkMode

Enable (true) continuous link mode. Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

bool enableHaltOnError

Enable (true) transfer halt on error. Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

bool enableRoundRobinArbitration

Enable (true) round robin channel arbitration method or fixed priority arbitration is used for channel selection

bool enableDebugMode

Enable(true) eDMA debug mode. When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete.

struct __edma__transfer__config

#include <fsl_edma.h> eDMA transfer configuration

This structure configures the source/destination transfer attribute.

Public Members

uint32_t srcAddr

Source data address.

uint32_t destAddr

Destination data address.

edma_transfer_size_t srcTransferSize

Source data transfer size.

edma_transfer_size_t destTransferSize

Destination data transfer size.

int16_t srcOffset

Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.

int16_t destOffset

Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

uint32_t minorLoopBytes

Bytes to transfer in a minor loop

uint32_t majorLoopCounts

Major loop iteration count.

struct _edma_channel_Preemption_config

#include <fsl_edma.h> eDMA channel priority configuration

Public Members

bool enableChannelPreemption

If true: a channel can be suspended by other channel with higher priority

bool enablePreemptAbility

If true: a channel can suspend other channel with low priority

uint8_t channelPriority

Channel priority

struct _edma_minor_offset_config

#include <fsl_edma.h> eDMA minor offset configuration

Public Members

bool enableSrcMinorOffset

Enable(true) or Disable(false) source minor loop offset.

bool enableDestMinorOffset

Enable(true) or Disable(false) destination minor loop offset.

uint32_t minorOffset

Offset for a minor loop mapping.

struct _edma_tcd

#include <fsl_edma.h> eDMA TCD.

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

Public Members

uint32_t SADDR

SADDR register, used to save source address

uint16_t SOFF
: SOFF register, save offset bytes every transfer

uint16_t ATTR
: ATTR register, source/destination transfer size and modulo

uint32_t NBYTES
: Nbytes register, minor loop length in bytes

uint32_t SLAST
: SLAST register

uint32_t DADDR
: DADDR register, used for destination address

uint16_t DOFF
: DOFF register, used for destination offset

uint16_t CITER
: CITER register, current minor loop numbers, for unfinished minor loop.

uint32_t DLAST_SGA
: DLASTSGA register, next tcd address used in scatter-gather mode

uint16_t CSR
: CSR register, for TCD control status

uint16_t BITER
: BITER register, begin minor loop count.

struct __edma_handle
: *#include <fsl_edma.h>* eDMA transfer handle structure

Public Members

edma_callback callback
: Callback function for major count exhausted.

void *userData
: Callback function parameter.

DMA_Type *base
: eDMA peripheral base address.

edma_tcd_t *tcdPool
: Pointer to memory stored TCDs.

uint8_t channel

eDMA channel number.

volatile int8_t header

The first TCD index. Should point to the next TCD to be loaded into the eDMA engine.

volatile int8_t tail

The last TCD index. Should point to the next TCD to be stored into the memory pool.

volatile int8_t tcdUsed

The number of used TCD slots. Should reflect the number of TCDs can be used/loaded in the memory.

volatile int8_t tcdSize

The total number of TCD slots in the queue.

uint8_t flags

The status of the current channel.

2.11 EWM: External Watchdog Monitor Driver

void EWM_Init(EWM_Type *base, const *ewm_config_t* *config)

Initializes the EWM peripheral.

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
ewm_config_t config;
EWM_GetDefaultConfig(&config);
config.compareHighValue = 0xAAU;
EWM_Init(ewm_base,&config);
```

Parameters

- base – EWM peripheral base address
- config – The configuration of the EWM

void EWM_Deinit(EWM_Type *base)

Deinitializes the EWM peripheral.

This function is used to shut down the EWM.

Parameters

- base – EWM peripheral base address

void EWM_GetDefaultConfig(*ewm_config_t* *config)

Initializes the EWM configuration structure.

This function initializes the EWM configuration structure to default values. The default values are as follows.

```
ewmConfig->enableEwm = true;
ewmConfig->enableEwmInput = false;
ewmConfig->setInputAssertLogic = false;
ewmConfig->enableInterrupt = false;
ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
ewmConfig->prescaler = 0;
ewmConfig->compareLowValue = 0;
ewmConfig->compareHighValue = 0xFEU;
```

See also:

ewm_config_t

Parameters

- config – Pointer to the EWM configuration structure.

static inline void EWM_EnableInterrupts(EWM_Type *base, uint32_t mask)

Enables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined
  - kEWM_InterruptEnable

static inline void EWM_DisableInterrupts(EWM_Type *base, uint32_t mask)

Disables the EWM interrupt.

This function enables the EWM interrupt.

Parameters

- base – EWM peripheral base address
- mask – The interrupts to disable The parameter can be combination of the following source if defined
  - kEWM_InterruptEnable

static inline uint32_t EWM_GetStatusFlags(EWM_Type *base)

Gets all status flags.

This function gets all status flags.

This is an example for getting the running flag.

```
uint32_t status;
status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
```

See also:

_ewm_status_flags_t

- True: a related status flag has been set.
- False: a related status flag is not set.

Parameters

- base – EWM peripheral base address

Returns
State of the status flag: asserted (true) or not-asserted (false).

void EWM_Refresh(EWM_Type *base)
Services the EWM.

This function resets the EWM counter to zero.

Parameters
- base – EWM peripheral base address

FSL_EWM_DRIVER_VERSION
EWM driver version 2.0.4.

enum _ewm_lpo_clock_source
Describes EWM clock source.

Values:

enumerator kEWM_LpoClockSource0
EWM clock sourced from lpo_clk[0]

enumerator kEWM_LpoClockSource1
EWM clock sourced from lpo_clk[1]

enumerator kEWM_LpoClockSource2
EWM clock sourced from lpo_clk[2]

enumerator kEWM_LpoClockSource3
EWM clock sourced from lpo_clk[3]

enum _ewm_interrupt_enable_t
EWM interrupt configuration structure with default settings all disabled.

This structure contains the settings for all of EWM interrupt configurations.

Values:

enumerator kEWM_InterruptEnable
Enable the EWM to generate an interrupt

enum _ewm_status_flags_t
EWM status flags.

This structure contains the constants for the EWM status flags for use in the EWM functions.

Values:

enumerator kEWM_RunningFlag
Running flag, set when EWM is enabled

typedef enum *_ewm_lpo_clock_source* ewm_lpo_clock_source_t
Describes EWM clock source.

typedef struct *_ewm_config* ewm_config_t

Data structure for EWM configuration.

This structure is used to configure the EWM.

struct _ewm_config

#include <fsl_ewm.h> Data structure for EWM configuration.

This structure is used to configure the EWM.

Public Members

bool enableEwm

Enable EWM module

bool enableEwmInput

Enable EWM_in input

bool setInputAssertLogic

EWM_in signal assertion state

bool enableInterrupt

Enable EWM interrupt

ewm_lpo_clock_source_t clockSource

Clock source select

uint8_t prescaler

Clock prescaler value

uint8_t compareLowValue

Compare low-register value

uint8_t compareHighValue

Compare high-register value

2.12 FGPIO Driver

2.13 C90TFS Flash Driver

2.14 FlexBus: External Bus Interface Driver

void FLEXBUS_Init(FB_Type *base, const *flexbus_config_t* *config)

Initializes and configures the FlexBus module.

This function enables the clock gate for FlexBus module. Only chip 0 is validated and set to known values. Other chips are disabled. Note that in this function, certain parameters, depending on external memories, must be set before using the *FLEXBUS_Init()* function.

This example shows how to set up the uart_state_t and the flexbus_config_t parameters and how to call the FLEXBUS_Init function by passing in these parameters.

```
flexbus_config_t flexbusConfig;
FLEXBUS_GetDefaultConfig(&flexbusConfig);
flexbusConfig.waitStates            = 2U;
flexbusConfig.chipBaseAddress       = 0x60000000U;
flexbusConfig.chipBaseAddressMask   = 7U;
FLEXBUS_Init(FB, &flexbusConfig);
```

Parameters

- base – FlexBus peripheral address.
- config – Pointer to the configuration structure

void FLEXBUS_Deinit(FB_Type *base)

De-initializes a FlexBus instance.

This function disables the clock gate of the FlexBus module clock.

Parameters

- base – FlexBus peripheral address.

void FLEXBUS_GetDefaultConfig(*flexbus_config_t* *config)

Initializes the FlexBus configuration structure.

This function initializes the FlexBus configuration structure to default value. The default values are.

```
fbConfig->chip                  = 0;
fbConfig->writeProtect          = 0;
fbConfig->burstWrite            = 0;
fbConfig->burstRead             = 0;
fbConfig->byteEnableMode        = 0;
fbConfig->autoAcknowledge       = true;
fbConfig->extendTransferAddress = 0;
fbConfig->secondaryWaitStates   = 0;
fbConfig->byteLaneShift         = kFLEXBUS_NotShifted;
fbConfig->writeAddressHold      = kFLEXBUS_Hold1Cycle;
fbConfig->readAddressHold       = kFLEXBUS_Hold1Or0Cycles;
fbConfig->addressSetup          = kFLEXBUS_FirstRisingEdge;
fbConfig->portSize              = kFLEXBUS_1Byte;
fbConfig->group1MultiplexControl = kFLEXBUS_MultiplexGroup1_FB_ALE;
fbConfig->group2MultiplexControl = kFLEXBUS_MultiplexGroup2_FB_CS4 ;
fbConfig->group3MultiplexControl = kFLEXBUS_MultiplexGroup3_FB_CS5;
fbConfig->group4MultiplexControl = kFLEXBUS_MultiplexGroup4_FB_TBST;
fbConfig->group5MultiplexControl = kFLEXBUS_MultiplexGroup5_FB_TA;
```

See also:

FLEXBUS_Init

Parameters

- config – Pointer to the initialization structure.

FSL_FLEXBUS_DRIVER_VERSION

Version 2.1.1.

enum _flexbus_port_size

Defines port size for FlexBus peripheral.

Values:

enumerator kFLEXBUS_4Bytes

32-bit port size

enumerator kFLEXBUS_1Byte

8-bit port size

enumerator kFLEXBUS_2Bytes

16-bit port size

enum _flexbus_write_address_hold

Defines number of cycles to hold address and attributes for FlexBus peripheral.

Values:

enumerator kFLEXBUS_Hold1Cycle

Hold address and attributes one cycles after FB_CSn negates on writes

enumerator kFLEXBUS_Hold2Cycles

Hold address and attributes two cycles after FB_CSn negates on writes

enumerator kFLEXBUS_Hold3Cycles

Hold address and attributes three cycles after FB_CSn negates on writes

enumerator kFLEXBUS_Hold4Cycles

Hold address and attributes four cycles after FB_CSn negates on writes

enum _flexbus_read_address_hold

Defines number of cycles to hold address and attributes for FlexBus peripheral.

Values:

enumerator kFLEXBUS_Hold1Or0Cycles

Hold address and attributes 1 or 0 cycles on reads

enumerator kFLEXBUS_Hold2Or1Cycles

Hold address and attributes 2 or 1 cycles on reads

enumerator kFLEXBUS_Hold3Or2Cycle

Hold address and attributes 3 or 2 cycles on reads

enumerator kFLEXBUS_Hold4Or3Cycle

Hold address and attributes 4 or 3 cycles on reads

enum _flexbus_address_setup

Address setup for FlexBus peripheral.

Values:

enumerator kFLEXBUS_FirstRisingEdge
Assert FB_CSn on first rising clock edge after address is asserted

enumerator kFLEXBUS_SecondRisingEdge
Assert FB_CSn on second rising clock edge after address is asserted

enumerator kFLEXBUS_ThirdRisingEdge
Assert FB_CSn on third rising clock edge after address is asserted

enumerator kFLEXBUS_FourthRisingEdge
Assert FB_CSn on fourth rising clock edge after address is asserted

enum _flexbus_bytelane_shift
Defines byte-lane shift for FlexBus peripheral.
Values:

enumerator kFLEXBUS_NotShifted
Not shifted. Data is left-justified on FB_AD

enumerator kFLEXBUS_Shifted
Shifted. Data is right justified on FB_AD

enum _flexbus_multiplex_group1_signal
Defines multiplex group1 valid signals.
Values:

enumerator kFLEXBUS_MultiplexGroup1_FB_ALE
FB_ALE

enumerator kFLEXBUS_MultiplexGroup1_FB_CS1
FB_CS1

enumerator kFLEXBUS_MultiplexGroup1_FB_TS
FB_TS

enum _flexbus_multiplex_group2_signal
Defines multiplex group2 valid signals.
Values:

enumerator kFLEXBUS_MultiplexGroup2_FB_CS4
FB_CS4

enumerator kFLEXBUS_MultiplexGroup2_FB_TSIZ0
FB_TSIZ0

enumerator kFLEXBUS_MultiplexGroup2_FB_BE_31_24
FB_BE_31_24

enum _flexbus_multiplex_group3_signal

Defines multiplex group3 valid signals.

Values:

enumerator kFLEXBUS_MultiplexGroup3_FB_CS5

FB_CS5

enumerator kFLEXBUS_MultiplexGroup3_FB_TSIZ1

FB_TSIZ1

enumerator kFLEXBUS_MultiplexGroup3_FB_BE_23_16

FB_BE_23_16

enum _flexbus_multiplex_group4_signal

Defines multiplex group4 valid signals.

Values:

enumerator kFLEXBUS_MultiplexGroup4_FB_TBST

FB_TBST

enumerator kFLEXBUS_MultiplexGroup4_FB_CS2

FB_CS2

enumerator kFLEXBUS_MultiplexGroup4_FB_BE_15_8

FB_BE_15_8

enum _flexbus_multiplex_group5_signal

Defines multiplex group5 valid signals.

Values:

enumerator kFLEXBUS_MultiplexGroup5_FB_TA

FB_TA

enumerator kFLEXBUS_MultiplexGroup5_FB_CS3

FB_CS3

enumerator kFLEXBUS_MultiplexGroup5_FB_BE_7_0

FB_BE_7_0

typedef enum *_flexbus_port_size* flexbus_port_size_t

Defines port size for FlexBus peripheral.

typedef enum *_flexbus_write_address_hold* flexbus_write_address_hold_t

Defines number of cycles to hold address and attributes for FlexBus peripheral.

typedef enum *_flexbus_read_address_hold* flexbus_read_address_hold_t

Defines number of cycles to hold address and attributes for FlexBus peripheral.

typedef enum *_flexbus_address_setup* flexbus_address_setup_t
Address setup for FlexBus peripheral.

typedef enum *_flexbus_bytelane_shift* flexbus_bytelane_shift_t
Defines byte-lane shift for FlexBus peripheral.

typedef enum *_flexbus_multiplex_group1_signal* flexbus_multiplex_group1_t
Defines multiplex group1 valid signals.

typedef enum *_flexbus_multiplex_group2_signal* flexbus_multiplex_group2_t
Defines multiplex group2 valid signals.

typedef enum *_flexbus_multiplex_group3_signal* flexbus_multiplex_group3_t
Defines multiplex group3 valid signals.

typedef enum *_flexbus_multiplex_group4_signal* flexbus_multiplex_group4_t
Defines multiplex group4 valid signals.

typedef enum *_flexbus_multiplex_group5_signal* flexbus_multiplex_group5_t
Defines multiplex group5 valid signals.

typedef struct *_flexbus_config* flexbus_config_t
Configuration structure that the user needs to set.

struct _flexbus_config
#include <fsl_flexbus.h> Configuration structure that the user needs to set.

Public Members

uint8_t chip
Chip FlexBus for validation

uint8_t waitStates
Value of wait states

uint8_t secondaryWaitStates
Value of secondary wait states

uint32_t chipBaseAddress
Chip base address for using FlexBus

uint32_t chipBaseAddressMask
Chip base address mask

bool writeProtect
Write protected

bool burstWrite
: Burst-Write enable

bool burstRead
: Burst-Read enable

bool byteEnableMode
: Byte-enable mode support

bool autoAcknowledge
: Auto acknowledge setting

bool extendTransferAddress
: Extend transfer start/extend address latch enable

bool secondaryWaitStatesEnable
: Enable secondary wait states

flexbus_port_size_t portSize
: Port size of transfer

flexbus_bytelane_shift_t byteLaneShift
: Byte-lane shift enable

flexbus_write_address_hold_t writeAddressHold
: Write address hold or deselect option

flexbus_read_address_hold_t readAddressHold
: Read address hold or deselect option

flexbus_address_setup_t addressSetup
: Address setup setting

flexbus_multiplex_group1_t group1MultiplexControl
: FlexBus Signal Group 1 Multiplex control

flexbus_multiplex_group2_t group2MultiplexControl
: FlexBus Signal Group 2 Multiplex control

flexbus_multiplex_group3_t group3MultiplexControl
: FlexBus Signal Group 3 Multiplex control

flexbus_multiplex_group4_t group4MultiplexControl
: FlexBus Signal Group 4 Multiplex control

flexbus_multiplex_group5_t group5MultiplexControl
: FlexBus Signal Group 5 Multiplex control

2.15 ftfx adapter

2.16 Ftftx CACHE Driver

enum _ftfx_cache_ram_func_constants

Constants for execute-in-RAM flash function.

Values:

enumerator kFTFx_CACHE_RamFuncMaxSizeInWords

The maximum size of execute-in-RAM function.

typedef struct *_flash_prefetch_speculation_status* ftfx_prefetch_speculation_status_t

FTFx prefetch speculation status.

typedef struct *_ftfx_cache_config* ftfx_cache_config_t

FTFx cache driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

status_t FTFx_CACHE_Init(*ftfx_cache_config_t* *config)

Initializes the global FTFx cache structure members.

This function checks and initializes the Flash module for the other FTFx cache APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.

status_t FTFx_CACHE_ClearCachePrefetchSpeculation(*ftfx_cache_config_t* *config, bool isPreProcess)

Process the cache/prefetch/speculation to the flash.

Parameters

- config – A pointer to the storage for the driver runtime state.
- isPreProcess – The possible option used to control flash cache/prefetch/speculation

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – Invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.

status_t FTFx_CACHE_PflashSetPrefetchSpeculation(*ftfx_prefetch_speculation_status_t* *speculationStatus)

Sets the PFlash prefetch speculation to the intended speculation status.

Parameters

- speculationStatus – The expected protect status to set to the PFlash protection register. Each bit is

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidSpeculationOption – An invalid speculation option argument is provided.

status_t FTFx_CACHE_PflashGetPrefetchSpeculation(*ftfx_prefetch_speculation_status_t* *speculationStatus)

Gets the PFlash prefetch speculation status.

Parameters

- speculationStatus – Speculation status returned by the PFlash IP.

Return values

kStatus_FTFx_Success – API was executed successfully.

struct _flash_prefetch_speculation_status

#include <fsl_ftfx_cache.h> FTFx prefetch speculation status.

Public Members

bool instructionOff

Instruction speculation.

bool dataOff

Data speculation.

union function_bit_operation_ptr_t

#include <fsl_ftfx_cache.h>

Public Members

uint32_t commadAddr

void (*callFlashCommand)(volatile uint32_t *base, uint32_t bitMask, uint32_t bitShift, uint32_t bitValue)

struct _ftfx_cache_config

#include <fsl_ftfx_cache.h> FTFx cache driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

Public Members

uint8_t flashMemoryIndex
0 - primary flash; 1 - secondary flash

function_bit_operation_ptr_t bitOperFuncAddr
An buffer point to the flash execute-in-RAM function.

2.17 ftfx controller

FTFx driver status codes.

Values:

enumerator kStatus_FTFx_Success
API is executed successfully

enumerator kStatus_FTFx_InvalidArgument
Invalid argument

enumerator kStatus_FTFx_SizeError
Error size

enumerator kStatus_FTFx_AlignmentError
Parameter is not aligned with the specified baseline

enumerator kStatus_FTFx_AddressError
Address is out of range

enumerator kStatus_FTFx_AccessError
Invalid instruction codes and out-of bound addresses

enumerator kStatus_FTFx_ProtectionViolation
The program/erase operation is requested to execute on protected areas

enumerator kStatus_FTFx_CommandFailure
Run-time error during command execution.

enumerator kStatus_FTFx_UnknownProperty
Unknown property.

enumerator kStatus_FTFx_EraseKeyError
API erase key is invalid.

enumerator kStatus_FTFx_RegionExecuteOnly
The current region is execute-only.

enumerator kStatus_FTFx_ExecuteInRamFunctionNotReady
Execute-in-RAM function is not available.

enumerator kStatus_FTFx_PartitionStatusUpdateFailure
Failed to update partition status.

enumerator kStatus_FTFx_SetFlexramAsEepromError
Failed to set FlexRAM as EEPROM.

enumerator kStatus_FTFx_RecoverFlexramAsRamError
Failed to recover FlexRAM as RAM.

enumerator kStatus_FTFx_SetFlexramAsRamError
Failed to set FlexRAM as RAM.

enumerator kStatus_FTFx_RecoverFlexramAsEepromError
Failed to recover FlexRAM as EEPROM.

enumerator kStatus_FTFx_CommandNotSupported
Flash API is not supported.

enumerator kStatus_FTFx_SwapSystemNotInUninitialized
Swap system is not in an uninitialzed state.

enumerator kStatus_FTFx_SwapIndicatorAddressError
The swap indicator address is invalid.

enumerator kStatus_FTFx_ReadOnlyProperty
The flash property is read-only.

enumerator kStatus_FTFx_InvalidPropertyValue
The flash property value is out of range.

enumerator kStatus_FTFx_InvalidSpeculationOption
The option of flash prefetch speculation is invalid.

enumerator kStatus_FTFx_CommandOperationInProgress
The option of flash command is processing.

enum _ftfx_driver_api_keys
Enumeration for FTFx driver API keys.

Note: The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

Values:

enumerator kFTFx_ApiEraseKey
Key value used to validate all FTFx erase APIs.

void FTFx_API_Init(*ftfx_config_t* *config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

status_t FTFx_API_UpdateFlexnvmPartitionStatus(*ftfx_config_t* *config)

Updates FlexNVM memory partition status according to data flash 0 IFR.

This function updates FlexNVM memory partition status.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FTFx_CMD_Erase(*ftfx_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the flash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.
- kStatus_FTFx_EraseKeyError – The API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_EraseSectorNonBlocking(*ftfx_config_t* *config, uint32_t start, uint32_t key)

Erases the flash sectors encompassed by parameters passed into function.

This function erases one flash sector size based on the start address.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.
- kStatus_FTFx_EraseKeyError – The API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.

status_t FTFx_CMD_EraseAll(*ftfx_config_t* *config, uint32_t key)

Erases entire flash.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FTFx_CMD_EraseAllUnsecure(*ftfx_config_t* *config, uint32_t key)

Erases the entire flash, including protected sectors.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FTFx_CMD_EraseAllExecuteOnlySegments(*ftfx_config_t* *config, uint32_t key)

Erases all program flash execute-only segments defined by the FXACC registers.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_Program(*ftfx_config_t* *config, uint32_t start, const uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_ProgramOnce(*ftfx_config_t* *config, uint32_t index, const uint8_t *src, uint32_t lengthInBytes)

Programs Program Once Field through parameters.

This function programs the Program Once Field with the desired data for a given flash area as determined by the index and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- index – The index indicating which area of the Program Once Field to be programmed.
- src – A pointer to the source buffer of data that is to be programmed into the Program Once Field.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_ProgramSection(*ftfx_config_t* *config, uint32_t start, const uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters via the Program Section command.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.

- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_SetFlexramAsRamError – Failed to set flexram as RAM.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_RecoverFlexramAsEepromError – Failed to recover FlexRAM as EEPROM.

status_t FTFx_CMD_ProgramPartition(*ftfx_config_t* *config, *ftfx_partition_flexram_load_opt_t* option, uint32_t eepromDataSizeCode, uint32_t flexnvmPartitionCode, uint8_t CSEcKeySize, uint8_t CFE)

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

Parameters

- config – Pointer to storage for the driver runtime state.
- option – The option used to set FlexRAM load behavior during reset.
- eepromDataSizeCode – Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
- flexnvmPartitionCode – Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – Invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.

- kStatus_FTFx_CommandFailure – Run-time error during command execution.

status_t FTFx_CMD_ReadOnce(*ftfx_config_t* *config, uint32_t index, uint8_t *dst, uint32_t lengthInBytes)

Reads the Program Once Field through parameters.

This function reads the read once feild with given index and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- index – The index indicating the area of program once field to be read.
- dst – A pointer to the destination buffer of data that is used to store data to be read.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_ReadResource(*ftfx_config_t* *config, uint32_t start, uint8_t *dst, uint32_t lengthInBytes, *ftfx_read_resource_opt_t* option)

Reads the resource with data at locations passed in through parameters.

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- dst – A pointer to the destination buffer of data that is used to store data to be read.
- lengthInBytes – The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
- option – The resource option which indicates which area should be read back.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with the specified baseline.

- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_VerifyErase(*ftfx_config_t* *config, uint32_t start, uint32_t lengthInBytes, *ftfx_margin_value_t* margin)

Verifies an erasure of the desired flash area at a specified margin level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_VerifyEraseAll(*ftfx_config_t* *config, *ftfx_margin_value_t* margin)

Verifies erasure of the entire flash at a specified margin level.

This function checks whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_VerifyEraseAllExecuteOnlySegments(*ftfx_config_t* *config, *ftfx_margin_value_t* margin)

Verifies whether the program flash execute-only segments have been erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_VerifyProgram(*ftfx_config_t* *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, *ftfx_margin_value_t* margin, uint32_t *failedAddress, uint32_t *failedData)

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. Must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- expectedData – A pointer to the expected data that is to be verified against.
- margin – Read margin choice.
- failedAddress – A pointer to the returned failing address.
- failedData – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_REG_GetSecurityState(*ftfx_config_t* *config, *ftfx_security_state_t* *state)

Returns the security state via the pointer passed into the function.

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

- config – A pointer to storage for the driver runtime state.
- state – A pointer to the value returned for the current security status code:

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

status_t FTFx_CMD_SecurityBypass(*ftfx_config_t* *config, const uint8_t *backdoorKey)

Allows users to bypass security with a backdoor key.

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

- config – A pointer to the storage for the driver runtime state.
- backdoorKey – A pointer to the user buffer containing the backdoor key.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_SetFlexramFunction(*ftfx_config_t* *config, *ftfx_flexram_func_opt_t* option)

Sets the FlexRAM function command.

Parameters

- config – A pointer to the storage for the driver runtime state.
- option – The option used to set the work mode of FlexRAM.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FTFx_CMD_SwapControl(*ftfx_config_t* *config, uint32_t address, *ftfx_swap_control_opt_t* option, *ftfx_swap_state_config_t* *returnInfo)

Configures the Swap function or checks the swap state of the Flash module.

Parameters

- config – A pointer to the storage for the driver runtime state.
- address – Address used to configure the flash Swap function.
- option – The possible option used to configure Flash Swap function or check the flash Swap status
- returnInfo – A pointer to the data which is used to return the information of flash Swap.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_SwapIndicatorAddressError – Swap indicator address is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

enum _ftfx_partition_flexram_load_option

Enumeration for the FlexRAM load during reset option.

Values:

enumerator kFTFx_PartitionFlexramLoadOptLoadedWithValidEepromData

FlexRAM is loaded with valid EEPROM data during reset sequence.

enumerator kFTFx_PartitionFlexramLoadOptNotLoaded

FlexRAM is not loaded during reset sequence.

enum _ftfx_read_resource_opt

Enumeration for the two possible options of flash read resource command.

Values:

enumerator kFTFx_ResourceOptionFlashIfr

Select code for Program flash 0 IFR, Program flash swap 0 IFR, Data flash 0 IFR

enumerator kFTFx_ResourceOptionVersionId

Select code for the version ID

enum _ftfx_margin_value

Enumeration for supported FTFx margin levels.

Values:

enumerator kFTFx_MarginValueNormal

Use the 'normal' read level for 1s.

enumerator kFTFx_MarginValueUser

Apply the 'User' margin to the normal read-1 level.

enumerator kFTFx_MarginValueFactory

Apply the 'Factory' margin to the normal read-1 level.

enumerator kFTFx_MarginValueInvalid

Not real margin level, Used to determine the range of valid margin level.

enum _ftfx_security_state

Enumeration for the three possible FTFx security states.

Values:

enumerator kFTFx_SecurityStateNotSecure

Flash is not secure.

enumerator kFTFx_SecurityStateBackdoorEnabled

Flash backdoor is enabled.

enumerator kFTFx_SecurityStateBackdoorDisabled

Flash backdoor is disabled.

enum _ftfx_flexram_function_option

Enumeration for the two possilbe options of set FlexRAM function command.

Values:

enumerator kFTFx_FlexramFuncOptAvailableAsRam

An option used to make FlexRAM available as RAM

enumerator kFTFx_FlexramFuncOptEepromQuickWriteRecovery

An option used to complete interrupted EEPROM quick write process

enumerator kFTFx_FlexramFuncOptEepromQuickWriteStatus

An option used to make EEPROM quick write status query

enumerator kFTFx_FlexramFuncOptAvailableForEepromQuickWrite

An option used to make FlexRAM available for EEPROM in Quick Write mode

enumerator kFTFx_FlexramFuncOptAvailableForEeprom

An option used to make FlexRAM available for EEPROM

enum _flash_acceleration_ram_property

Enumeration for acceleration ram property.

Values:

enumerator kFLASH_AccelerationRamSize

enum _ftfx_swap_control_option

Enumeration for the possible options of Swap control commands.

Values:

enumerator kFTFx_SwapControlOptionIntializeSystem

An option used to initialize the Swap system

enumerator kFTFx_SwapControlOptionSetInUpdateState

An option used to set the Swap in an update state

enumerator kFTFx_SwapControlOptionSetInCompleteState

An option used to set the Swap in a complete state

enumerator kFTFx_SwapControlOptionReportStatus

An option used to report the Swap status

enumerator kFTFx_SwapControlOptionDisableSystem

An option used to disable the Swap status

enum _ftfx_swap_state

Enumeration for the possible flash Swap status.

Values:

enumerator kFTFx_SwapStateUninitialized
Flash Swap system is in an uninitialized state.

enumerator kFTFx_SwapStateReady
Flash Swap system is in a ready state.

enumerator kFTFx_SwapStateUpdate
Flash Swap system is in an update state.

enumerator kFTFx_SwapStateUpdateErased
Flash Swap system is in an updateErased state.

enumerator kFTFx_SwapStateComplete
Flash Swap system is in a complete state.

enumerator kFTFx_SwapStateDisabled
Flash Swap system is in a disabled state.

enum _ftfx_swap_block_status
Enumeration for the possible flash Swap block status.
Values:

enumerator kFTFx_SwapBlockStatusLowerHalfProgramBlocksAtZero
Swap block status is that lower half program block at zero.

enumerator kFTFx_SwapBlockStatusUpperHalfProgramBlocksAtZero
Swap block status is that upper half program block at zero.

enum _ftfx_memory_type
Enumeration for FTFx memory type.
Values:

enumerator kFTFx_MemTypePflash

enumerator kFTFx_MemTypeFlexnvm

typedef enum *_ftfx_partition_flexram_load_option* ftfx_partition_flexram_load_opt_t
Enumeration for the FlexRAM load during reset option.

typedef enum *_ftfx_read_resource_opt* ftfx_read_resource_opt_t
Enumeration for the two possible options of flash read resource command.

typedef enum *_ftfx_margin_value* ftfx_margin_value_t
Enumeration for supported FTFx margin levels.

typedef enum *_ftfx_security_state* ftfx_security_state_t
Enumeration for the three possible FTFx security states.

typedef enum *_ftfx_flexram_function_option* ftfx_flexram_func_opt_t

Enumeration for the two possilbe options of set FlexRAM function command.

typedef enum *_ftfx_swap_control_option* ftfx_swap_control_opt_t

Enumeration for the possible options of Swap control commands.

typedef enum *_ftfx_swap_state* ftfx_swap_state_t

Enumeration for the possible flash Swap status.

typedef enum *_ftfx_swap_block_status* ftfx_swap_block_status_t

Enumeration for the possible flash Swap block status.

typedef struct *_ftfx_swap_state_config* ftfx_swap_state_config_t

Flash Swap information.

typedef struct *_ftfx_special_mem* ftfx_spec_mem_t

ftfx special memory access information.

typedef struct *_ftfx_mem_descriptor* ftfx_mem_desc_t

Flash memory descriptor.

typedef struct *_ftfx_ops_config* ftfx_ops_config_t

Active FTFx information for the current operation.

typedef struct *_ftfx_ifr_descriptor* ftfx_ifr_desc_t

Flash IFR memory descriptor.

typedef struct *_ftfx_config* ftfx_config_t

Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

struct __ftfx_swap_state_config

#include <fsl_ftfx_controller.h> Flash Swap information.

Public Members

ftfx_swap_state_t flashSwapState

The current Swap system status.

ftfx_swap_block_status_t currentSwapBlockStatus

The current Swap block status.

ftfx_swap_block_status_t nextSwapBlockStatus

The next Swap block status.

struct __ftfx_special_mem

#include <fsl_ftfx_controller.h> ftfx special memory access information.

Public Members

uint32_t base
Base address of flash special memory.

uint32_t size
size of flash special memory.

uint32_t count
flash special memory count.

struct __ftfx_mem_descriptor
#include <fsl_ftfx_controller.h> Flash memory descriptor.

Public Members

uint32_t blockBase
A base address of the flash block

uint32_t aliasBlockBase
A base address of the alias flash block

uint32_t totalSize
The size of the flash block.

uint32_t sectorSize
The size in bytes of a sector of flash.

uint32_t blockCount
A number of flash blocks.

struct __ftfx_ops_config
#include <fsl_ftfx_controller.h> Active FTFx information for the current operation.

Public Members

uint32_t convertedAddress
A converted address for the current flash type.

struct __ftfx_ifr_descriptor
#include <fsl_ftfx_controller.h> Flash IFR memory descriptor.

union function_ptr_t
#include <fsl_ftfx_controller.h>

Public Members

uint32_t commadAddr

void (*callFlashCommand)(volatile uint8_t *FTMRx_fstat)

struct _ftfx_config

#include <fsl_ftfx_controller.h> Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

Public Members

uint32_t flexramBlockBase

The base address of the FlexRAM/acceleration RAM

uint32_t flexramTotalSize

The size of the FlexRAM/acceleration RAM

uint16_t eepromTotalSize

The size of EEPROM area which was partitioned from FlexRAM

function_ptr_t runCmdFuncAddr

An buffer point to the flash execute-in-RAM function.

struct ___unnamed11___

Public Members

uint8_t type

Type of flash block.

uint8_t index

Index of flash block.

struct feature

struct addrAligment

struct feature

struct resRange

Public Members

uint8_t versionIdStart
: Version ID start address

uint32_t pflashIfrStart
: Program Flash 0 IFR start address

uint32_t dflashIfrStart
: Data Flash 0 IFR start address

uint32_t pflashSwapIfrStart
: Program Flash Swap IFR start address

struct idxInfo

2.18 ftfx feature

FTFx_DRIVER_IS_FLASH_RESIDENT
: Flash driver location.

  Used for the flash resident application.

FTFx_DRIVER_IS_EXPORTED
: Flash Driver Export option.

  Used for the MCUXpresso SDK application.

FTFx_FLASH1_HAS_PROT_CONTROL
: Indicates whether the secondary flash has its own protection register in flash module.

FTFx_FLASH1_HAS_XACC_CONTROL
: Indicates whether the secondary flash has its own Execute-Only access register in flash module.

FTFx_DRIVER_HAS_FLASH1_SUPPORT
: Indicates whether the secondary flash is supported in the Flash driver.

FTFx_FLASH_COUNT

FTFx_FLASH1_IS_INDEPENDENT_BLOCK

2.19 Ftftx FLASH Driver

status_t FLASH_Init(*flash_config_t* *config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FLASH_Erase(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the Dflash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully; the appropriate number of flash sectors based on the desired start address and length were erased successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.
- kStatus_FTFx_EraseKeyError – The API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_EraseSectorNonBlocking(*flash_config_t* *config, uint32_t start, uint32_t key)

Erases the Dflash sectors encompassed by parameters passed into function.

This function erases one flash sector size based on the start address, and it is executed asynchronously.

NOTE: This function can only erase one flash sector at a time, and the other commands can be executed after the previous command has been completed.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.
- kStatus_FTFx_EraseKeyError – The API erase key is invalid.

status_t FLASH_EraseAll(*flash_config_t* *config, uint32_t key)

Erases entire flexnvm.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully; the all pflash and flexnvm were erased successfully, the swap and eeprom have been reset to unconfigured state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FLASH_EraseAllUnsecure(*flash_config_t* *config, uint32_t key)

Erases the entire flexnvm, including protected sectors.

Parameters

- config – Pointer to the storage for the driver runtime state.

- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully; the protected sectors of flash were reset to unprotected status.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FLASH_Program(*flash_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desired data were programed successfully into flash based on desired start address and length.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_ProgramOnce(*flash_config_t* *config, uint32_t index, uint8_t *src, uint32_t lengthInBytes)

Program the Program-Once-Field through parameters.

This function Program the Program-once-feild with given index and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- index – The index indicating the area of program once field to be read.
- src – A pointer to the source buffer of data that is used to store data to be write.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; The index indicating the area of program once field was programed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_ProgramSection(*flash_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters via the Program Section command.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desired data have been programed successfully into flash based on start address and length.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.

- kStatus_FTFx_SetFlexramAsRamError – Failed to set flexram as RAM.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_RecoverFlexramAsEepromError – Failed to recover FlexRAM as EEPROM.

status_t FLASH_ReadResource(*flash_config_t* *config, uint32_t start, uint8_t *dst, uint32_t lengthInBytes, *ftfx_read_resource_opt_t* option)

Reads the resource with data at locations passed in through parameters.

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- dst – A pointer to the destination buffer of data that is used to store data to be read.
- lengthInBytes – The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
- option – The resource option which indicates which area should be read back.

Return values

- kStatus_FTFx_Success – API was executed successfully; the data have been read successfully from program flash IFR, data flash IFR space, and the Version ID field.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_ReadOnce(*flash_config_t* *config, uint32_t index, uint8_t *dst, uint32_t lengthInBytes)

Reads the Program Once Field through parameters.

This function reads the read once feild with given index and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- index – The index indicating the area of program once field to be read.
- dst – A pointer to the destination buffer of data that is used to store data to be read.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; the data have been successfuly read form Program flash0 IFR map and Program Once field based on index and length.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_VerifyErase(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, *ftfx_margin_value_t* margin)

Verifies an erasure of the desired flash area at a specified margin level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully; the specified FLASH region has been erased.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.

- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_VerifyEraseAll(*flash_config_t* *config, *ftfx_margin_value_t* margin)

Verifies erasure of the entire flash at a specified margin level.

This function checks whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully; all program flash and flexnvm were in erased state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_VerifyProgram(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, *ftfx_margin_value_t* margin, uint32_t *failedAddress, uint32_t *failedData)

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programmed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. Must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- expectedData – A pointer to the expected data that is to be verified against.
- margin – Read margin choice.
- failedAddress – A pointer to the returned failing address.
- failedData – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desired data have been successfully programed into specified FLASH region.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.

- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_GetSecurityState(*flash_config_t* *config, *ftfx_security_state_t* *state)

Returns the security state via the pointer passed into the function.

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

- config – A pointer to storage for the driver runtime state.
- state – A pointer to the value returned for the current security status code:

Return values

- kStatus_FTFx_Success – API was executed successfully; the security state of flash was stored to state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

status_t FLASH_SecurityBypass(*flash_config_t* *config, const uint8_t *backdoorKey)

Allows users to bypass security with a backdoor key.

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

- config – A pointer to the storage for the driver runtime state.
- backdoorKey – A pointer to the user buffer containing the backdoor key.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_SetFlexramFunction(*flash_config_t* *config, *ftfx_flexram_func_opt_t* option)

Sets the FlexRAM function command.

Parameters

- config – A pointer to the storage for the driver runtime state.
- option – The option used to set the work mode of FlexRAM.

Return values

- kStatus_FTFx_Success – API was executed successfully; the FlexRAM has been successfully configured as RAM or EEPROM.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLASH_Swap(*flash_config_t* *config, uint32_t address, bool isSetEnable)

Swaps the lower half flash with the higher half flash.

Parameters

- config – A pointer to the storage for the driver runtime state.
- address – Address used to configure the flash swap function
- isSetEnable – The possible option used to configure the Flash Swap function or check the flash Swap status.

Return values

- kStatus_FTFx_Success – API was executed successfully; the lower half flash and higher half flash have been swaped.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_SwapIndicatorAddressError – Swap indicator address is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_SwapSystemNotInUninitialized – Swap system is not in an uninitialized state.

status_t FLASH_IsProtected(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, *flash_prot_state_t* *protection_state)

Returns the protection state of the desired flash area via the pointer passed into the function.

This function retrieves the current flash protect status for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.

- start – The start address of the desired flash memory to be checked. Must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be checked. Must be word-aligned.
- protection_state – A pointer to the value returned for the current protection status code for the desired flash area.

Return values

- kStatus_FTFx_Success – API was executed successfully; the protection state of specified FLASH region was stored to protection_state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.

status_t FLASH_IsExecuteOnly(*flash_config_t* *config, uint32_t start, uint32_t lengthInBytes, *flash_xacc_state_t* *access_state)

Returns the access state of the desired flash area via the pointer passed into the function.

This function retrieves the current flash access status for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be checked. Must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be checked. Must be word-aligned.
- access_state – A pointer to the value returned for the current access status code for the desired flash area.

Return values

- kStatus_FTFx_Success – API was executed successfully; the executeOnly state of specified FLASH region was stored to access_state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – The parameter is not aligned to the specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.

status_t FLASH_PflashSetProtection(*flash_config_t* *config, *pflash_prot_status_t* *protectStatus)

Sets the PFlash Protection to the intended protection status.

Parameters

- config – A pointer to storage for the driver runtime state.
- protectStatus – The expected protect status to set to the PFlash protection register. Each bit is corresponding to protection of 1/32(64) of the total PFlash. The least significant bit is corresponding to the lowest address area of PFlash. The most significant bit is corresponding to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

- kStatus_FTFx_Success – API was executed successfully; the specified FLASH region is protected.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.

status_t FLASH_PflashGetProtection(*flash_config_t* *config, *pflash_prot_status_t* *protectStatus)

Gets the PFlash protection status.

Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – Protect status returned by the PFlash IP. Each bit is corresponding to the protection of 1/32(64) of the total PFlash. The least significant bit corresponds to the lowest address area of the PFlash. The most significant bit corresponds to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

- kStatus_FTFx_Success – API was executed successfully; the Protection state was stored to protectStatus;
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

status_t FLASH_GetProperty(*flash_config_t* *config, *flash_property_tag_t* whichProperty, uint32_t *value)

Returns the desired flash property.

Parameters

- config – A pointer to the storage for the driver runtime state.
- whichProperty – The desired property from the list of properties in enum flash_property_tag_t
- value – A pointer to the value returned for the desired flash property.

Return values

- kStatus_FTFx_Success – API was executed successfully; the flash property was stored to value.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_UnknownProperty – An unknown property tag.

status_t FLASH_GetCommandState(void)

Get previous command status.

This function is used to obtain the execution status of the previous command.

Return values

- kStatus_FTFx_Success – The previous command is executed successfully.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.

- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

FSL_FLASH_DRIVER_VERSION

Flash driver version for SDK.

Version 3.3.0.

FSL_FLASH_DRIVER_VERSION_ROM

Flash driver version for ROM.

Version 3.0.0.

enum _flash_protection_state

Enumeration for the three possible flash protection levels.

Values:

enumerator kFLASH_ProtectionStateUnprotected

Flash region is not protected.

enumerator kFLASH_ProtectionStateProtected

Flash region is protected.

enumerator kFLASH_ProtectionStateMixed

Flash is mixed with protected and unprotected region.

enum _flash_execute_only_access_state

Enumeration for the three possible flash execute access levels.

Values:

enumerator kFLASH_AccessStateUnLimited

Flash region is unlimited.

enumerator kFLASH_AccessStateExecuteOnly

Flash region is execute only.

enumerator kFLASH_AccessStateMixed

Flash is mixed with unlimited and execute only region.

enum _flash_property_tag

Enumeration for various flash properties.

Values:

enumerator kFLASH_PropertyPflash0SectorSize

Pflash sector size property.

enumerator kFLASH_PropertyPflash0TotalSize

Pflash total size property.

enumerator kFLASH_PropertyPflash0BlockSize
Pflash block size property.

enumerator kFLASH_PropertyPflash0BlockCount
Pflash block count property.

enumerator kFLASH_PropertyPflash0BlockBaseAddr
Pflash block base address property.

enumerator kFLASH_PropertyPflash0FacSupport
Pflash fac support property.

enumerator kFLASH_PropertyPflash0AccessSegmentSize
Pflash access segment size property.

enumerator kFLASH_PropertyPflash0AccessSegmentCount
Pflash access segment count property.

enumerator kFLASH_PropertyPflash1SectorSize
Pflash sector size property.

enumerator kFLASH_PropertyPflash1TotalSize
Pflash total size property.

enumerator kFLASH_PropertyPflash1BlockSize
Pflash block size property.

enumerator kFLASH_PropertyPflash1BlockCount
Pflash block count property.

enumerator kFLASH_PropertyPflash1BlockBaseAddr
Pflash block base address property.

enumerator kFLASH_PropertyPflash1FacSupport
Pflash fac support property.

enumerator kFLASH_PropertyPflash1AccessSegmentSize
Pflash access segment size property.

enumerator kFLASH_PropertyPflash1AccessSegmentCount
Pflash access segment count property.

enumerator kFLASH_PropertyFlexRamBlockBaseAddr
FlexRam block base address property.

enumerator kFLASH_PropertyFlexRamTotalSize
FlexRam total size property.

typedef enum *_flash_protection_state* flash_prot_state_t

Enumeration for the three possible flash protection levels.

typedef union *_pflash_protection_status* pflash_prot_status_t

PFlash protection status.

typedef enum *_flash_execute_only_access_state* flash_xacc_state_t

Enumeration for the three possible flash execute access levels.

typedef enum *_flash_property_tag* flash_property_tag_t

Enumeration for various flash properties.

typedef struct *_flash_config* flash_config_t

Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

kStatus_FLASH_Success

kFLASH_ApiEraseKey

union _pflash_protection_status

#include <fsl_ftfx_flash.h> PFlash protection status.

Public Members

uint32_t protl

PROT[31:0] .

uint32_t proth

PROT[63:32].

uint8_t protsl

PROTS[7:0] .

uint8_t protsh

PROTS[15:8] .

uint8_t reserved[2]

struct _flash_config

#include <fsl_ftfx_flash.h> Flash driver state information.

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

2.20 Ftftx FLEXNVM Driver

status_t FLEXNVM_Init(*flexnvm_config_t* *config)

Initializes the global flash properties structure members.

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

- config – Pointer to the storage for the driver runtime state.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FLEXNVM_DflashErase(*flexnvm_config_t* *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)

Erases the Dflash sectors encompassed by parameters passed into function.

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

- config – The pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
- key – The value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully; the appropriate number of date flash sectors based on the desired start address and length were erased successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – The parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – The address is out of range.
- kStatus_FTFx_EraseKeyError – The API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_EraseAll(*flexnvm_config_t* *config, uint32_t key)

Erases entire flexnvm.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully; the entire flexnvm has been erased successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FLEXNVM_EraseAllUnsecure(*flexnvm_config_t* *config, uint32_t key)

Erases the entire flexnvm, including protected sectors.

Parameters

- config – Pointer to the storage for the driver runtime state.
- key – A value used to validate all flash erase APIs.

Return values

- kStatus_FTFx_Success – API was executed successfully; the flexnvm is not in securityi state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_EraseKeyError – API erase key is invalid.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_PartitionStatusUpdateFailure – Failed to update the partition status.

status_t FLEXNVM_DflashProgram(*flexnvm_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desired date have been successfully programed into specified date flash region.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with the specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_DflashProgramSection(*flexnvm_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs flash with data at locations passed in through parameters via the Program Section command.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desired date have been successfully programed into specified date flash area.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.

- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_SetFlexramAsRamError – Failed to set flexram as RAM.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.
- kStatus_FTFx_RecoverFlexramAsEepromError – Failed to recover FlexRAM as EEPROM.

status_t FLEXNVM_ProgramPartition(*flexnvm_config_t* *config, *ftfx_partition_flexram_load_opt_t* option, uint32_t eepromDataSizeCode, uint32_t flexnvmPartitionCode)

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

Parameters

- config – Pointer to storage for the driver runtime state.
- option – The option used to set FlexRAM load behavior during reset.
- eepromDataSizeCode – Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
- flexnvmPartitionCode – Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.

Return values

- kStatus_FTFx_Success – API was executed successfully; the FlexNVM block for use as data flash, EEPROM backup, or a combination of both have been Prepared.
- kStatus_FTFx_InvalidArgument – Invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.

status_t FLEXNVM_ProgramPartition_CSE(*flexnvm_config_t* *config, *ftfx_partition_flexram_load_opt_t* option, uint32_t eepromDataSizeCode, uint32_t flexnvmPartitionCode, uint8_t CSEcKeySize, uint8_t SFE)

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM. This is the CSE enabled version for IP's like FTFC.

Parameters

- config – Pointer to storage for the driver runtime state.
- option – The option used to set FlexRAM load behavior during reset.
- eepromDataSizeCode – Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
- flexnvmPartitionCode – Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.
- CSEcKeySize – CSEc/SHE key size, see RM for details and possible values
- SFE – Security Flag Extension (SFE), see RM for details and possible values

Return values

- kStatus_FTFx_Success – API was executed successfully; the FlexNVM block for use as data flash, EEPROM backup, or a combination of both have been Prepared.
- kStatus_FTFx_InvalidArgument – Invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.

status_t FLEXNVM_ReadResource(*flexnvm_config_t* *config, uint32_t start, uint8_t *dst, uint32_t lengthInBytes, *ftfx_read_resource_opt_t* option)

Reads the resource with data at locations passed in through parameters.

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- dst – A pointer to the destination buffer of data that is used to store data to be read.
- lengthInBytes – The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
- option – The resource option which indicates which area should be read back.

Return values

- kStatus_FTFx_Success – API was executed successfully; the data have been read successfully from program flash IFR, data flash IFR space, and the Version ID field
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with the specified baseline.

- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_DflashVerifyErase(*flexnvm_config_t* *config, uint32_t start, uint32_t lengthInBytes, *ftfx_margin_value_t* margin)

Verifies an erasure of the desired flash area at a specified margin level.

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully; the specified data flash region is in erased state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_VerifyEraseAll(*flexnvm_config_t* *config, *ftfx_margin_value_t* margin)

Verifies erasure of the entire flash at a specified margin level.

This function checks whether the flash is erased to the specified read margin level.

Parameters

- config – A pointer to the storage for the driver runtime state.
- margin – Read margin choice.

Return values

- kStatus_FTFx_Success – API was executed successfully; the entire flexnvm region is in erased state.

- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_DflashVerifyProgram(*flexnvm_config_t* *config, uint32_t start, uint32_t lengthInBytes, const uint8_t *expectedData, *ftfx_margin_value_t* margin, uint32_t *failedAddress, uint32_t *failedData)

Verifies programming of the desired flash area at a specified margin level.

This function verifies the data programmed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be verified. Must be word-aligned.
- lengthInBytes – The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
- expectedData – A pointer to the expected data that is to be verified against.
- margin – Read margin choice.
- failedAddress – A pointer to the returned failing address.
- failedData – A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desired data hve been programed successfully into specified data flash region.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AlignmentError – Parameter is not aligned with specified baseline.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_GetSecurityState(*flexnvm_config_t* *config, *ftfx_security_state_t* *state)

Returns the security state via the pointer passed into the function.

This function retrieves the current flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

- config – A pointer to storage for the driver runtime state.
- state – A pointer to the value returned for the current security status code:

Return values

- kStatus_FTFx_Success – API was executed successfully; the security state of flexnvm was stored to state.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.

status_t FLEXNVM_SecurityBypass(*flexnvm_config_t* *config, const uint8_t *backdoorKey)

Allows users to bypass security with a backdoor key.

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

- config – A pointer to the storage for the driver runtime state.
- backdoorKey – A pointer to the user buffer containing the backdoor key.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.
- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_SetFlexramFunction(*flexnvm_config_t* *config, *ftfx_flexram_func_opt_t* option)

Sets the FlexRAM function command.

Parameters

- config – A pointer to the storage for the driver runtime state.
- option – The option used to set the work mode of FlexRAM.

Return values

- kStatus_FTFx_Success – API was executed successfully; the FlexRAM has been successfully configured as RAM or EEPROM
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_ExecuteInRamFunctionNotReady – Execute-in-RAM function is not available.
- kStatus_FTFx_AccessError – Invalid instruction codes and out-of bounds addresses.

- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_CommandFailure – Run-time error during the command execution.

status_t FLEXNVM_DflashSetProtection(*flexnvm_config_t* *config, uint8_t protectStatus)

Sets the DFlash protection to the intended protection status.

Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – The expected protect status to set to the DFlash protection register. Each bit corresponds to the protection of the 1/8 of the total DFlash. The least significant bit corresponds to the lowest address area of the DFlash. The most significant bit corresponds to the highest address area of the DFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

- kStatus_FTFx_Success – API was executed successfully; the specified DFlash region is protected.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_CommandNotSupported – Flash API is not supported.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.

status_t FLEXNVM_DflashGetProtection(*flexnvm_config_t* *config, uint8_t *protectStatus)

Gets the DFlash protection status.

Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – DFlash Protect status returned by the PFlash IP. Each bit corresponds to the protection of the 1/8 of the total DFlash. The least significant bit corresponds to the lowest address area of the DFlash. The most significant bit corresponds to the highest address area of the DFlash, and so on. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_CommandNotSupported – Flash API is not supported.

status_t FLEXNVM_EepromSetProtection(*flexnvm_config_t* *config, uint8_t protectStatus)

Sets the EEPROM protection to the intended protection status.

Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – The expected protect status to set to the EEPROM protection register. Each bit corresponds to the protection of the 1/8 of the total EEPROM. The least significant bit corresponds to the lowest address area of the EEPROM. The most significant bit corresponds to the highest address area of EEPROM, and so on. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_CommandNotSupported – Flash API is not supported.
- kStatus_FTFx_CommandFailure – Run-time error during command execution.

status_t FLEXNVM_EepromGetProtection(*flexnvm_config_t* *config, uint8_t *protectStatus)

Gets the EEPROM protection status.

Parameters

- config – A pointer to the storage for the driver runtime state.
- protectStatus – DFlash Protect status returned by the PFlash IP. Each bit corresponds to the protection of the 1/8 of the total EEPROM. The least significant bit corresponds to the lowest address area of the EEPROM. The most significant bit corresponds to the highest address area of the EEPROM. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_CommandNotSupported – Flash API is not supported.

status_t FLEXNVM_GetProperty(*flexnvm_config_t* *config, *flexnvm_property_tag_t* whichProperty, uint32_t *value)

Returns the desired flexnvm property.

Parameters

- config – A pointer to the storage for the driver runtime state.
- whichProperty – The desired property from the list of properties in enum flexnvm_property_tag_t
- value – A pointer to the value returned for the desired flexnvm property.

Return values

- kStatus_FTFx_Success – API was executed successfully.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_UnknownProperty – An unknown property tag.

enum _flexnvm_property_tag

Enumeration for various flexnvm properties.

Values:

enumerator kFLEXNVM_PropertyDflashSectorSize

Dflash sector size property.

enumerator kFLEXNVM_PropertyDflashTotalSize

Dflash total size property.

enumerator kFLEXNVM_PropertyDflashBlockSize

Dflash block size property.

enumerator kFLEXNVM_PropertyDflashBlockCount

Dflash block count property.

enumerator kFLEXNVM_PropertyDflashBlockBaseAddr

Dflash block base address property.

enumerator kFLEXNVM_PropertyAliasDflashBlockBaseAddr

Dflash block base address Alias property.

enumerator kFLEXNVM_PropertyFlexRamBlockBaseAddr

FlexRam block base address property.

enumerator kFLEXNVM_PropertyFlexRamTotalSize

FlexRam total size property.

enumerator kFLEXNVM_PropertyEepromTotalSize

EEPROM total size property.

typedef enum *_flexnvm_property_tag* flexnvm_property_tag_t

Enumeration for various flexnvm properties.

typedef struct *_flexnvm_config* flexnvm_config_t

Flexnvm driver state information.

An instance of this structure is allocated by the user of the Flexnvm driver and passed into each of the driver APIs.

status_t FLEXNVM_EepromWrite(*flexnvm_config_t* *config, uint32_t start, uint8_t *src, uint32_t lengthInBytes)

Programs the EEPROM with data at locations passed in through parameters.

This function programs the emulated EEPROM with the desired data for a given flash area as determined by the start address and length.

Parameters

- config – A pointer to the storage for the driver runtime state.
- start – The start address of the desired flash memory to be programmed. Must be word-aligned.
- src – A pointer to the source buffer of data that is to be programmed into the flash.
- lengthInBytes – The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

- kStatus_FTFx_Success – API was executed successfully; the desires data have been successfully programed into specified eeprom region.
- kStatus_FTFx_InvalidArgument – An invalid argument is provided.
- kStatus_FTFx_AddressError – Address is out of range.
- kStatus_FTFx_SetFlexramAsEepromError – Failed to set flexram as eeprom.

- kStatus_FTFx_ProtectionViolation – The program/erase operation is requested to execute on protected areas.
- kStatus_FTFx_RecoverFlexramAsRamError – Failed to recover the FlexRAM as RAM.

struct _flexnvm_config

#include <fsl_ftfx_flexnvm.h> Flexnvm driver state information.

An instance of this structure is allocated by the user of the Flexnvm driver and passed into each of the driver APIs.

2.21 ftfx utilities

ALIGN_DOWN(x, a)

Alignment(down) utility.

ALIGN_UP(x, a)

Alignment(up) utility.

MAKE_VERSION(major, minor, bugfix)

Constructs the version number for drivers.

MAKE_STATUS(group, code)

Constructs a status code value from a group and a code number.

FOUR_CHAR_CODE(a, b, c, d)

Constructs the four character code for the Flash driver API key.

B1P4(b)

bytes2word utility.

B1P3(b)

B1P2(b)

B1P1(b)

B2P3(b)

B2P2(b)

B2P1(b)

B3P2(b)

B3P1(b)

BYTE2WORD_1_3(x, y)

BYTE2WORD_2_2(x, y)

BYTE2WORD_3_1(x, y)

BYTE2WORD_1_1_2(x, y, z)

BYTE2WORD_1_2_1(x, y, z)

BYTE2WORD_2_1_1(x, y, z)

BYTE2WORD_1_1_1_1(x, y, z, w)

2.22 FTM: FlexTimer Driver

status_t FTM_Init(FTM_Type *base, const *ftm_config_t* *config)

Ungates the FTM clock and configures the peripheral for basic operation.

Note: This API should be called at the beginning of the application which is using the FTM driver. If the FTM instance has only TPM features, please use the TPM driver.

Parameters

- base – FTM peripheral base address
- config – Pointer to the user configuration structure.

Returns

kStatus_Success indicates success; Else indicates failure.

void FTM_Deinit(FTM_Type *base)

Gates the FTM clock.

Parameters

- base – FTM peripheral base address

void FTM_GetDefaultConfig(*ftm_config_t* *config)

Fills in the FTM configuration structure with the default settings.

The default values are:

```
config->prescale = kFTM_Prescale_Divide_1;
config->bdmMode = kFTM_BdmMode_0;
config->pwmSyncMode = kFTM_SoftwareTrigger;
config->reloadPoints = 0;
config->faultMode = kFTM_Fault_Disable;
config->faultFilterValue = 0;
config->deadTimePrescale = kFTM_Deadtime_Prescale_1;
config->deadTimeValue =  0;
config->extTriggers = 0;
config->chnlInitState = 0;
config->chnlPolarity = 0;
config->useGlobalTimeBase = false;
config->hwTriggerResetCount = false;
config->swTriggerResetCount = true;
```

Parameters

- config – Pointer to the user configuration structure.

static inline *ftm_clock_prescale_t* FTM_CalculateCounterClkDiv(FTM_Type *base, uint32_t counterPeriod_Hz, uint32_t srcClock_Hz)

brief Calculates the counter clock prescaler.

This function calculates the values for SC[PS] bit.

param base FTM peripheral base address param counterPeriod_Hz The desired frequency in Hz which corresponding to the time when the counter reaches the mod value param srcClock_Hz FTM counter clock in Hz

return Calculated clock prescaler value, see *ftm_clock_prescale_t*.

status_t FTM_SetupPwm(FTM_Type *base, const *ftm_chnl_pwm_signal_param_t* *chnlParams, uint8_t numOfChnls, *ftm_pwm_mode_t* mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)

Configures the PWM signal parameters.

Call this function to configure the PWM signal period, mode, duty cycle, and edge. Use this function to configure all FTM channels that are used to output a PWM signal.

Parameters

- base – FTM peripheral base address
- chnlParams – Array of PWM channel parameters to configure the channel(s)
- numOfChnls – Number of channels to configure; This should be the size of the array passed in
- mode – PWM operation mode, options available in enumeration *ftm_pwm_mode_t*
- pwmFreq_Hz – PWM signal frequency in Hz
- srcClock_Hz – FTM counter clock in Hz

Returns

kStatus_Success if the PWM setup was successful kStatus_Error on failure

status_t FTM_UpdatePwmDutycycle(FTM_Type *base, *ftm_chnl_t* chnlNumber, *ftm_pwm_mode_t* currentPwmMode, uint8_t dutyCyclePercent)

Updates the duty cycle of an active PWM signal.

Note: This function only writes the new duty cycle value to the CnV buffer register. *FTM_Init()* unconditionally enables enhanced PWM synchronization mode by setting SYNCONF.SYNCMODE = 1 and COMBINE.SYNCENn = 1 in FTM_SetPwmSync(), regardless of the configuration passed by the user. Therefore, the buffered CnV value does NOT take effect immediately. The caller must trigger a register reload after this function returns, using one of the following methods:

- Software trigger: call FTM_SetSoftwareTrigger(base, true) (if pwmSyncMode includes kFTM_SoftwareTrigger). Note that if swTriggerResetCount was set to true during initialization, this trigger will also force the FTM counter to the CNTIN value (SYNCONF.SWRSTCNT = 1).
- Hardware trigger: no software call is needed. If pwmSyncMode includes kFTM_HardwareTrigger_0/1/2, the reload happens automatically when the corresponding hardware trigger fires (SYNCONF.HWWRBUF is set by *FTM_Init()*).
- LDOK: call FTM_SetLdok(base, true). The new value becomes active at the next reload point: by default at counter overflow (MOD to CNTIN), or at earlier events if additional reload points are configured via FTM_SetReloadPoints().

Parameters

- base – FTM peripheral base address
- chnlNumber – The channel/channel pair number. In combined mode, this represents the channel pair number
- currentPwmMode – The current PWM mode set during PWM setup
- dutyCyclePercent – New PWM pulse width; The value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)

Returns
kStatus_Success if the PWM update was successful kStatus_Error on failure

void FTM_UpdateChnlEdgeLevelSelect(FTM_Type *base, *ftm_chnl_t* chnlNumber, uint8_t level)

Updates the edge level selection for a channel.

Parameters

- base – FTM peripheral base address
- chnlNumber – The channel number
- level – The level to be set to the ELSnB:ELSnA field; Valid values are 00, 01, 10, 11. See the Kinetis SoC reference manual for details about this field.

status_t FTM_SetupPwmMode(FTM_Type *base, const *ftm_chnl_pwm_config_param_t* *chnlParams, uint8_t numOfChnls, *ftm_pwm_mode_t* mode)

Configures the PWM mode parameters.

Call this function to configure the PWM signal mode, duty cycle in ticks, and edge. Use this function to configure all FTM channels that are used to output a PWM signal. Please note that: This API is similar with *FTM_SetupPwm()* API, but will not set the timer period, and this API will set channel match value in timer ticks, not period percent.

Parameters

- base – FTM peripheral base address
- chnlParams – Array of PWM channel parameters to configure the channel(s)
- numOfChnls – Number of channels to configure; This should be the size of the array passed in
- mode – PWM operation mode, options available in enumeration *ftm_pwm_mode_t*

Returns
kStatus_Success if the PWM setup was successful kStatus_Error on failure

void FTM_ConfigSinglePWM(FTM_Type *base, const *ftm_chnl_param_t* *chnlParams, *ftm_chnl_t* chnlNumber)

Configure FTM edge aligned PWM or center aligned PWM by each channel.

This function configure PWM signal by setting channel n value register. Need to invoke FTM_SetInitialModuloValue to configure FTM period.

Parameters

- base – FTM peripheral base address
- chnlParams – PWM configuration structure pointer.
- chnlPairNumber – Channel number.

void FTM_ConfigCombinePWM(FTM_Type *base, const *ftm_chnl_param_t* *chnlParams, *ftm_chnl_t* chnlPairNumber)

Configure FTM Combine PWM, Modified Combine PWM or Asymmetrical PWM by each channel pair.

This function configure PWM signal by setting channel n value register. Need to invoke FTM_SetInitialModuloValue to configure FTM period.

Parameters

- base – FTM peripheral base address
- chnlParams – PWM configuration structure pointer.

- chnlPairNumber – Channel pair number, options are 0, 1, 2, 3.

void FTM_SetupInputCapture(FTM_Type *base, *ftm_chnl_t* chnlNumber, *ftm_input_capture_edge_t* captureMode, uint32_t filterValue)

Enables capturing an input signal on the channel using the function parameters.

When the edge specified in the captureMode argument occurs on the channel, the FTM counter is captured into the CnV register. The user has to read the CnV register separately to get this value. The filter function is disabled if the filterVal argument passed in is 0. The filter function is available only for channels 0, 1, 2, 3.

Parameters

- base – FTM peripheral base address
- chnlNumber – The channel number
- captureMode – Specifies which edge to capture
- filterValue – Filter value, specify 0 to disable filter. Available only for channels 0-3.

void FTM_SetupOutputCompare(FTM_Type *base, *ftm_chnl_t* chnlNumber, *ftm_output_compare_mode_t* compareMode, uint32_t compareValue)

Configures the FTM to generate timed pulses.

When the FTM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

Parameters

- base – FTM peripheral base address
- chnlNumber – The channel number
- compareMode – Action to take on the channel output when the compare condition is met
- compareValue – Value to be programmed in the CnV register.

void FTM_SetupDualEdgeCapture(FTM_Type *base, *ftm_chnl_t* chnlPairNumber, const *ftm_dual_edge_capture_param_t* *edgeParam, uint32_t filterValue)

Configures the dual edge capture mode of the FTM.

This function sets up the dual edge capture mode on a channel pair. The capture edge for the channel pair and the capture mode (one-shot or continuous) is specified in the parameter argument. The filter function is disabled if the filterVal argument passed is zero. The filter function is available only on channels 0 and 2. The user has to read the channel CnV registers separately to get the capture values.

Parameters

- base – FTM peripheral base address
- chnlPairNumber – The FTM channel pair number; options are 0, 1, 2, 3
- edgeParam – Sets up the dual edge capture function
- filterValue – Filter value, specify 0 to disable filter. Available only for channel pair 0 and 1.

void FTM_EnableInterrupts(FTM_Type *base, uint32_t mask)

Enables the selected FTM interrupts.

Parameters

- base – FTM peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *ftm_interrupt_enable_t*

void FTM_DisableInterrupts(FTM_Type *base, uint32_t mask)

Disables the selected FTM interrupts.

Parameters

- base – FTM peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *ftm_interrupt_enable_t*

uint32_t FTM_GetEnabledInterrupts(FTM_Type *base)

Gets the enabled FTM interrupts.

Parameters

- base – FTM peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *ftm_interrupt_enable_t*

uint32_t FTM_GetInstance(FTM_Type *base)

Gets the instance from the base address.

Parameters

- base – FTM peripheral base address

Returns

The FTM instance

uint32_t FTM_GetStatusFlags(FTM_Type *base)

Gets the FTM status flags.

Parameters

- base – FTM peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration *ftm_status_flags_t*

void FTM_ClearStatusFlags(FTM_Type *base, uint32_t mask)

Clears the FTM status flags.

Parameters

- base – FTM peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration *ftm_status_flags_t*

static inline void FTM_SetTimerPeriod(FTM_Type *base, uint32_t ticks)

Sets the timer period in units of ticks.

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

Note:

a. This API allows the user to use the FTM module as a timer. Do not mix usage of this API with FTM's PWM setup API's.

b. Call the utility macros provided in the fsl_common.h to convert usec or msec to ticks.

Parameters

- base – FTM peripheral base address
- ticks – A timer period in units of ticks, which should be equal or greater than 1.

static inline void FTM_SetInitialModuloValue(FTM_Type *base, uint16_t initialValue, uint16_t moduloValue)

Set initial value and modulo value for FTM.

Parameters

- base – FTM peripheral base address
- initialValue – FTM counter initial value.
- moduloValue – FTM counter modulo value.

static inline uint32_t FTM_GetCurrentTimerCount(FTM_Type *base)

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: Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

- base – FTM peripheral base address

Returns

The current counter value in ticks

static inline void FTM_SetChannelMatchValue(FTM_Type *base, *ftm_chnl_t* chnlNumber, uint16_t value)

Set channel match value for output.

Parameters

- base – FTM peripheral base address
- chnlNumber – Channel to set.
- value – Channel match value for output.

static inline uint32_t FTM_GetInputCaptureValue(FTM_Type *base, *ftm_chnl_t* chnlNumber)

Reads the captured value.

This function returns the captured value of a FTM channel configured in input capture or dual edge capture mode.

Note: Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

- base – FTM peripheral base address
- chnlNumber – Channel to be read

Returns

The captured FTM counter value of the input modes.

static inline void FTM_StartTimer(FTM_Type *base, *ftm_clock_source_t* clockSource)

Starts the FTM counter.

Parameters

- base – FTM peripheral base address
- clockSource – FTM clock source; After the clock source is set, the counter starts running.

static inline void FTM_StopTimer(FTM_Type *base)

Stops the FTM counter.

Parameters

- base – FTM peripheral base address

static inline uint32_t FTM_GetSoftwareOutputValue(FTM_Type *base)

Get channel software output status.

Parameters

- base – FTM peripheral base address

Returns

Status of channel software output, logical OR value of *ftm_channel_index_t*.

static inline uint32_t FTM_GetSoftwareOutputEnable(FTM_Type *base)

Get channel software enable status.

Parameters

- base – FTM peripheral base address

Returns

Status of channel software enable, logical OR value of *ftm_channel_index_t*.

static inline void FTM_SetSoftwareOutputCtrl(FTM_Type *base, uint32_t chnlEnable, uint32_t chnlValue)

Enables or disables the channel software output control and set channel software output value.

Parameters

- base – FTM peripheral base address
- chnlEnable – Channels to enable or disable software output control, logical OR of enumeration *ftm_channel_index_t* members.
- chnlValue – Channels output value, logical OR of enumeration *ftm_channel_index_t* members

static inline void FTM_SetSoftwareCtrlEnable(FTM_Type *base, *ftm_chnl_t* chnlNumber, bool value)

Enables or disables the channel software output control.

Parameters

- base – FTM peripheral base address
- chnlNumber – Channel to be enabled or disabled
- value – true: channel output is affected by software output control false: channel output is unaffected by software output control

static inline void FTM_SetSoftwareCtrlVal(FTM_Type *base, *ftm_chnl_t* chnlNumber, bool value)

Sets the channel software output control value.

Parameters

- base – FTM peripheral base address.
- chnlNumber – Channel to be configured
- value – true to set 1, false to set 0

static inline void FTM_SetFaultControlEnable(FTM_Type *base, *ftm_chnl_t* chnlPairNumber, bool value)

This function enables/disables the fault control in a channel pair.

Parameters

- base – FTM peripheral base address
- chnlPairNumber – The FTM channel pair number; options are 0, 1, 2, 3
- value – true: Enable fault control for this channel pair; false: No fault control

static inline void FTM_SetDeadTimeEnable(FTM_Type *base, *ftm_chnl_t* chnlPairNumber, bool value)

This function enables/disables the dead time insertion in a channel pair.

Parameters

- base – FTM peripheral base address
- chnlPairNumber – The FTM channel pair number; options are 0, 1, 2, 3
- value – true: Insert dead time in this channel pair; false: No dead time inserted

static inline void FTM_SetComplementaryEnable(FTM_Type *base, *ftm_chnl_t* chnlPairNumber, bool value)

This function enables/disables complementary mode in a channel pair.

Parameters

- base – FTM peripheral base address
- chnlPairNumber – The FTM channel pair number; options are 0, 1, 2, 3
- value – true: enable complementary mode; false: disable complementary mode

static inline void FTM_SetInvertEnable(FTM_Type *base, *ftm_chnl_t* chnlPairNumber, bool value)

This function enables/disables inverting control in a channel pair.

Parameters

- base – FTM peripheral base address
- chnlPairNumber – The FTM channel pair number; options are 0, 1, 2, 3
- value – true: enable inverting; false: disable inverting

void FTM_SetupQuadDecode(FTM_Type *base, const *ftm_phase_params_t* *phaseAParams, const *ftm_phase_params_t* *phaseBParams, *ftm_quad_decode_mode_t* quadMode)

Configures the parameters and activates the quadrature decoder mode.

Parameters

- base – FTM peripheral base address
- phaseAParams – Phase A configuration parameters
- phaseBParams – Phase B configuration parameters

- quadMode – Selects encoding mode used in quadrature decoder mode

static inline void FTM_SetQuadDecoderModuloValue(FTM_Type *base, uint32_t startValue, uint32_t overValue)

Sets the modulo values for Quad Decoder.

The modulo values configure the minimum and maximum values that the Quad decoder counter can reach. After the counter goes over, the counter value goes to the other side and decrease/increase again.

Parameters

- base – FTM peripheral base address.
- startValue – The low limit value for Quad Decoder counter.
- overValue – The high limit value for Quad Decoder counter.

static inline uint32_t FTM_GetQuadDecoderCounterValue(FTM_Type *base)

Gets the current Quad Decoder counter value.

Parameters

- base – FTM peripheral base address.

Returns

Current quad Decoder counter value.

static inline void FTM_ClearQuadDecoderCounterValue(FTM_Type *base)

Clears the current Quad Decoder counter value.

The counter is set as the initial value.

Parameters

- base – FTM peripheral base address.

FSL_FTM_DRIVER_VERSION

FTM driver version 2.7.6.

enum _ftm_chnl

List of FTM channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kFTM_Chnl_0

FTM channel number 0

enumerator kFTM_Chnl_1

FTM channel number 1

enumerator kFTM_Chnl_2

FTM channel number 2

enumerator kFTM_Chnl_3

FTM channel number 3

enumerator kFTM_Chnl_4
FTM channel number 4

enumerator kFTM_Chnl_5
FTM channel number 5

enumerator kFTM_Chnl_6
FTM channel number 6

enumerator kFTM_Chnl_7
FTM channel number 7

enum _ftm_fault_input
List of FTM faults.
Values:

enumerator kFTM_Fault_0
FTM fault 0 input pin

enumerator kFTM_Fault_1
FTM fault 1 input pin

enumerator kFTM_Fault_2
FTM fault 2 input pin

enumerator kFTM_Fault_3
FTM fault 3 input pin

enum _ftm_pwm_mode
FTM PWM operation modes.
Values:

enumerator kFTM_EdgeAlignedPwm
Edge-aligned PWM

enumerator kFTM_CenterAlignedPwm
Center-aligned PWM

enumerator kFTM_EdgeAlignedCombinedPwm
Edge-aligned combined PWM

enumerator kFTM_CenterAlignedCombinedPwm
Center-aligned combined PWM

enumerator kFTM_AsymmetricalCombinedPwm
Asymmetrical combined PWM

enum _ftm_pwm_level_select

FTM PWM output pulse mode: high-true, low-true or no output.

Note: kFTM_NoPwmSignal: ELSnB:ELSnA = 0:0 kFTM_LowTrue: ELSnB:ELSnA = 0:1 EPWM: Channel n output is forced low at counter overflow, forced high at channel n match. CPWM: Channel n output is forced low at channel n match when counting down, and forced high at channel n match when counting up. Combined PWM: Channel n output is forced high at beginning of period and at channel n+1 match. It is forced low at the channel n match. kFTM_HighTrue: ELSnB:ELSnA = 1:0 EPWM: Channel n output is forced high at counter overflow, forced low at channel n match. CPWM: Channel n output is forced high at channel n match when counting down, and forced low at channel n match when counting up. Combined PWM: Channel n output is forced low at beginning of period and at channel n+1 match. It is forced high at the channel n match.

Values:

enumerator kFTM_NoPwmSignal

No PWM output on pin

enumerator kFTM_LowTrue

Low true pulses

enumerator kFTM_HighTrue

High true pulses

enum _ftm_output_compare_mode

FlexTimer output compare mode.

Values:

enumerator kFTM_NoOutputSignal

No channel output when counter reaches CnV

enumerator kFTM_ToggleOnMatch

Toggle output

enumerator kFTM_ClearOnMatch

Clear output

enumerator kFTM_SetOnMatch

Set output

enum _ftm_input_capture_edge

FlexTimer input capture edge.

Values:

enumerator kFTM_RisingEdge

Capture on rising edge only

enumerator kFTM_FallingEdge
Capture on falling edge only

enumerator kFTM_RiseAndFallEdge
Capture on rising or falling edge

enum _ftm_dual_edge_capture_mode
FlexTimer dual edge capture modes.
Values:

enumerator kFTM_OneShot
One-shot capture mode

enumerator kFTM_Continuous
Continuous capture mode

enum _ftm_quad_decode_mode
FlexTimer quadrature decode modes.
Values:

enumerator kFTM_QuadPhaseEncode
Phase A and Phase B encoding mode

enumerator kFTM_QuadCountAndDir
Count and direction encoding mode

enum _ftm_phase_polarity
FlexTimer quadrature phase polarities.
Values:

enumerator kFTM_QuadPhaseNormal
Phase input signal is not inverted

enumerator kFTM_QuadPhaseInvert
Phase input signal is inverted

enum _ftm_deadtime_prescale
FlexTimer pre-scaler factor for the dead time insertion.
Values:

enumerator kFTM_Deadtime_Prescale_1
Divide by 1

enumerator kFTM_Deadtime_Prescale_4
Divide by 4

enumerator kFTM_Deadtime_Prescale_16
Divide by 16

enum _ftm_clock_source

FlexTimer clock source selection.

Values:

enumerator kFTM_SystemClock

System clock selected

enumerator kFTM_FixedClock

Fixed frequency clock

enumerator kFTM_ExternalClock

External clock

enum _ftm_clock_prescale

FlexTimer pre-scaler factor selection for the clock source.

Values:

enumerator kFTM_Prescale_Divide_1

Divide by 1

enumerator kFTM_Prescale_Divide_2

Divide by 2

enumerator kFTM_Prescale_Divide_4

Divide by 4

enumerator kFTM_Prescale_Divide_8

Divide by 8

enumerator kFTM_Prescale_Divide_16

Divide by 16

enumerator kFTM_Prescale_Divide_32

Divide by 32

enumerator kFTM_Prescale_Divide_64

Divide by 64

enumerator kFTM_Prescale_Divide_128

Divide by 128

enum _ftm_bdm_mode

Options for the FlexTimer behaviour in BDM Mode.

Values:

enumerator kFTM_BdmMode_0

FTM counter stopped, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers

enumerator kFTM_BdmMode_1

FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are forced to their safe value , writes to MOD,CNTIN and C(n)V registers bypass the register buffers

enumerator kFTM_BdmMode_2

FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are frozen when chip enters in BDM mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers

enumerator kFTM_BdmMode_3

FTM counter in functional mode, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers is in fully functional mode

enum _ftm_fault_mode

Options for the FTM fault control mode.

Values:

enumerator kFTM_Fault_Disable

Fault control is disabled for all channels

enumerator kFTM_Fault_EvenChnls

Enabled for even channels only(0,2,4,6) with manual fault clearing

enumerator kFTM_Fault_AllChnlsMan

Enabled for all channels with manual fault clearing

enumerator kFTM_Fault_AllChnlsAuto

Enabled for all channels with automatic fault clearing

enum _ftm_external_trigger

FTM external trigger options.

Note: Actual available external trigger sources are SoC-specific

Values:

enumerator kFTM_Chnl0Trigger

Generate trigger when counter equals chnl 0 CnV reg

enumerator kFTM_Chnl1Trigger

Generate trigger when counter equals chnl 1 CnV reg

enumerator kFTM_Chnl2Trigger

Generate trigger when counter equals chnl 2 CnV reg

enumerator kFTM_Chnl3Trigger

Generate trigger when counter equals chnl 3 CnV reg

enumerator kFTM_Chnl4Trigger
Generate trigger when counter equals chnl 4 CnV reg

enumerator kFTM_Chnl5Trigger
Generate trigger when counter equals chnl 5 CnV reg

enumerator kFTM_Chnl6Trigger
Available on certain SoC's, generate trigger when counter equals chnl 6 CnV reg

enumerator kFTM_Chnl7Trigger
Available on certain SoC's, generate trigger when counter equals chnl 7 CnV reg

enumerator kFTM_InitTrigger
Generate Trigger when counter is updated with CNTIN

enumerator kFTM_ReloadInitTrigger
Available on certain SoC's, trigger on reload point

enum _ftm_pwm_sync_method
FlexTimer PWM sync options to update registers with buffer.
Values:

enumerator kFTM_SoftwareTrigger
Software triggers PWM sync

enumerator kFTM_HardwareTrigger_0
Hardware trigger 0 causes PWM sync

enumerator kFTM_HardwareTrigger_1
Hardware trigger 1 causes PWM sync

enumerator kFTM_HardwareTrigger_2
Hardware trigger 2 causes PWM sync

enum _ftm_reload_point
FTM options available as loading point for register reload.

Note: Actual available reload points are SoC-specific

Values:

enumerator kFTM_Chnl0Match
Channel 0 match included as a reload point

enumerator kFTM_Chnl1Match
Channel 1 match included as a reload point

enumerator kFTM_Chnl2Match
Channel 2 match included as a reload point

enumerator kFTM_Chnl3Match
Channel 3 match included as a reload point

enumerator kFTM_Chnl4Match
Channel 4 match included as a reload point

enumerator kFTM_Chnl5Match
Channel 5 match included as a reload point

enumerator kFTM_Chnl6Match
Channel 6 match included as a reload point

enumerator kFTM_Chnl7Match
Channel 7 match included as a reload point

enumerator kFTM_CntMax
Use in up-down count mode only, reload when counter reaches the maximum value

enumerator kFTM_CntMin
Use in up-down count mode only, reload when counter reaches the minimum value

enumerator kFTM_HalfCycMatch
Available on certain SoC's, half cycle match reload point

enum _ftm_interrupt_enable
List of FTM interrupts.

Note: Actual available interrupts are SoC-specific

Values:

enumerator kFTM_Chnl0InterruptEnable
Channel 0 interrupt

enumerator kFTM_Chnl1InterruptEnable
Channel 1 interrupt

enumerator kFTM_Chnl2InterruptEnable
Channel 2 interrupt

enumerator kFTM_Chnl3InterruptEnable
Channel 3 interrupt

enumerator kFTM_Chnl4InterruptEnable
Channel 4 interrupt

enumerator kFTM_Chnl5InterruptEnable
Channel 5 interrupt

enumerator kFTM_Chnl6InterruptEnable
Channel 6 interrupt

enumerator kFTM_Chnl7InterruptEnable
Channel 7 interrupt

enumerator kFTM_FaultInterruptEnable
Fault interrupt

enumerator kFTM_TimeOverflowInterruptEnable
Time overflow interrupt

enumerator kFTM_ReloadInterruptEnable
Reload interrupt; Available only on certain SoC's

enum _ftm_status_flags
List of FTM flags.

Note: Actual available flags are SoC-specific

Values:

enumerator kFTM_Chnl0Flag
Channel 0 Flag

enumerator kFTM_Chnl1Flag
Channel 1 Flag

enumerator kFTM_Chnl2Flag
Channel 2 Flag

enumerator kFTM_Chnl3Flag
Channel 3 Flag

enumerator kFTM_Chnl4Flag
Channel 4 Flag

enumerator kFTM_Chnl5Flag
Channel 5 Flag

enumerator kFTM_Chnl6Flag
Channel 6 Flag

enumerator kFTM_Chnl7Flag
Channel 7 Flag

enumerator kFTM_FaultFlag
Fault Flag

enumerator kFTM_TimeOverflowFlag
Time overflow Flag

enumerator kFTM_ChnlTriggerFlag
Channel trigger Flag

enumerator kFTM_ReloadFlag
Reload Flag; Available only on certain SoC's

enum _ftm_channel_index
List of FTM channel index used in logic OR.
Values:

enumerator kFTM_Chnl0_Mask
Channel 0 Mask

enumerator kFTM_Chnl1_Mask
Channel 1 Mask

enumerator kFTM_Chnl2_Mask
Channel 2 Mask

enumerator kFTM_Chnl3_Mask
Channel 3 Mask

enumerator kFTM_Chnl4_Mask
Channel 4 Mask

enumerator kFTM_Chnl5_Mask
Channel 5 Mask

enumerator kFTM_Chnl6_Mask
Channel 6 Mask

enumerator kFTM_Chnl7_Mask
Channel 7 Mask

typedef enum *_ftm_chnl* ftm_chnl_t
List of FTM channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_ftm_fault_input* ftm_fault_input_t
List of FTM faults.

typedef enum *_ftm_pwm_mode* ftm_pwm_mode_t

FTM PWM operation modes.

typedef enum *_ftm_pwm_level_select* ftm_pwm_level_select_t

FTM PWM output pulse mode: high-true, low-true or no output.

Note: kFTM_NoPwmSignal: ELSnB:ELSnA = 0:0 kFTM_LowTrue: ELSnB:ELSnA = 0:1 EPWM: Channel n output is forced low at counter overflow, forced high at channel n match. CPWM: Channel n output is forced low at channel n match when counting down, and forced high at channel n match when counting up. Combined PWM: Channel n output is forced high at beginning of period and at channel n+1 match. It is forced low at the channel n match. kFTM_HighTrue: ELSnB:ELSnA = 1:0 EPWM: Channel n output is forced high at counter overflow, forced low at channel n match. CPWM: Channel n output is forced high at channel n match when counting down, and forced low at channel n match when counting up. Combined PWM: Channel n output is forced low at beginning of period and at channel n+1 match. It is forced high at the channel n match.

typedef struct *_ftm_chnl_pwm_signal_param* ftm_chnl_pwm_signal_param_t

Options to configure a FTM channel's PWM signal.

typedef struct *_ftm_chnl_pwm_config_param* ftm_chnl_pwm_config_param_t

Options to configure a FTM channel using precise setting.

typedef struct *_ftm_chnl_param* ftm_chnl_param_t

General options to configure a FTM channel using precise setting.

typedef enum *_ftm_output_compare_mode* ftm_output_compare_mode_t

FlexTimer output compare mode.

typedef enum *_ftm_input_capture_edge* ftm_input_capture_edge_t

FlexTimer input capture edge.

typedef enum *_ftm_dual_edge_capture_mode* ftm_dual_edge_capture_mode_t

FlexTimer dual edge capture modes.

typedef struct *_ftm_dual_edge_capture_param* ftm_dual_edge_capture_param_t

FlexTimer dual edge capture parameters.

typedef enum *_ftm_quad_decode_mode* ftm_quad_decode_mode_t

FlexTimer quadrature decode modes.

typedef enum *_ftm_phase_polarity* ftm_phase_polarity_t

FlexTimer quadrature phase polarities.

typedef struct *_ftm_phase_param* ftm_phase_params_t

FlexTimer quadrature decode phase parameters.

typedef struct *_ftm_fault_param* ftm_fault_param_t

Structure is used to hold the parameters to configure a FTM fault.

typedef enum *_ftm_deadtime_prescale* ftm_deadtime_prescale_t
FlexTimer pre-scaler factor for the dead time insertion.

typedef enum *_ftm_clock_source* ftm_clock_source_t
FlexTimer clock source selection.

typedef enum *_ftm_clock_prescale* ftm_clock_prescale_t
FlexTimer pre-scaler factor selection for the clock source.

typedef enum *_ftm_bdm_mode* ftm_bdm_mode_t
Options for the FlexTimer behaviour in BDM Mode.

typedef enum *_ftm_fault_mode* ftm_fault_mode_t
Options for the FTM fault control mode.

typedef enum *_ftm_external_trigger* ftm_external_trigger_t
FTM external trigger options.

Note: Actual available external trigger sources are SoC-specific

typedef enum *_ftm_pwm_sync_method* ftm_pwm_sync_method_t
FlexTimer PWM sync options to update registers with buffer.

typedef enum *_ftm_reload_point* ftm_reload_point_t
FTM options available as loading point for register reload.

Note: Actual available reload points are SoC-specific

typedef enum *_ftm_interrupt_enable* ftm_interrupt_enable_t
List of FTM interrupts.

Note: Actual available interrupts are SoC-specific

typedef enum *_ftm_status_flags* ftm_status_flags_t
List of FTM flags.

Note: Actual available flags are SoC-specific

typedef enum *_ftm_channel_index* ftm_channel_index_t
List of FTM channel index used in logic OR.

typedef struct *_ftm_config* ftm_config_t
FTM configuration structure.

This structure holds the configuration settings for the FTM peripheral. To initialize this structure to reasonable defaults, call the *FTM_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 FTM_SetupFaultInput(FTM_Type *base, *ftm_fault_input_t* faultNumber, const *ftm_fault_param_t* *faultParams)

Sets up the working of the FTM fault inputs protection.

FTM can have up to 4 fault inputs. This function sets up fault parameters, fault level, and input filter.

Parameters

- base – FTM peripheral base address
- faultNumber – FTM fault to configure.
- faultParams – Parameters passed in to set up the fault

static inline void FTM_SetGlobalTimeBaseOutputEnable(FTM_Type *base, bool enable)

Enables or disables the FTM global time base signal generation to other FTMs.

Parameters

- base – FTM peripheral base address
- enable – true to enable, false to disable

static inline void FTM_SetOutputMask(FTM_Type *base, *ftm_chnl_t* chnlNumber, bool mask)

Sets the FTM peripheral timer channel output mask.

Parameters

- base – FTM peripheral base address
- chnlNumber – Channel to be configured
- mask – true: masked, channel is forced to its inactive state; false: unmasked

static inline void FTM_SetPwmOutputEnable(FTM_Type *base, *ftm_chnl_t* chnlNumber, bool value)

Allows users to enable an output on an FTM channel.

To enable the PWM channel output call this function with val=true. For input mode, call this function with val=false.

Parameters

- base – FTM peripheral base address
- chnlNumber – Channel to be configured
- value – true: enable output; false: output is disabled, used in input mode

static inline void FTM_SetSoftwareTrigger(FTM_Type *base, bool enable)

Enables or disables the FTM software trigger for PWM synchronization.

Parameters

- base – FTM peripheral base address
- enable – true: software trigger is selected, false: software trigger is not selected

static inline void FTM_SetWriteProtection(FTM_Type *base, bool enable)

Enables or disables the FTM write protection.

Parameters

- base – FTM peripheral base address
- enable – true: Write-protection is enabled, false: Write-protection is disabled

static inline void FTM_EnableDmaTransfer(FTM_Type *base, *ftm_chnl_t* chnlNumber, bool enable)

Enable DMA transfer or not.

Note: CHnIE bit needs to be set when calling this API. The channel DMA transfer request is generated and the channel interrupt is not generated if (CHnF = 1) when DMA and CHnIE bits are set.

Parameters

- base – FTM peripheral base address.
- chnlNumber – Channel to be configured
- enable – true to enable, false to disable

static inline void FTM_SetLdok(FTM_Type *base, bool value)

Enable the LDOK bit.

This function enables loading updated values.

Parameters

- base – FTM peripheral base address
- value – true: loading updated values is enabled; false: loading updated values is disabled.

static inline void FTM_SetHalfCycReloadMatchValue(FTM_Type *base, uint32_t ticks)

Sets the half cycle relade period in units of ticks.

This function can be callled to set the half-cycle reload value when half-cycle matching is enabled as a reload point. Note: Need enable kFTM_HalfCycMatch as reload point, and when this API call after *FTM_StartTimer()*, the new HCR value will not be active until next reload point (need call FTM_SetLdok to set LDOK) or register synchronization.

Parameters

- base – FTM peripheral base address
- ticks – A timer period in units of ticks, which should be equal or greater than 1.

static inline void FTM_SetLoadFreq(FTM_Type *base, uint32_t loadfreq)

Set load frequency value.

Parameters

- base – FTM peripheral base address.
- loadfreq – PWM reload frequency, range: 0 ~ 31.

struct __ftm_chnl_pwm_signal_param

#include <fsl_ftm.h> Options to configure a FTM channel's PWM signal.

Public Members

ftm_chnl_t chnlNumber

The channel/channel pair number. In combined mode, this represents the channel pair number.

ftm_pwm_level_select_t level

PWM output active level select.

uint8_t dutyCyclePercent

PWM pulse width, value should be between 0 to 100 0 = inactive signal(0% duty cycle)... 100 = always active signal (100% duty cycle).

uint8_t firstEdgeDelayPercent

Used only in kFTM_AsymmetricalCombinedPwm mode to generate an asymmetrical PWM. Specifies the delay to the first edge in a PWM period. If unsure leave as 0; Should be specified as a percentage of the PWM period

bool enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

bool enableDeadtime

Used only in combined PWM mode with enable complementary. true: The deadtime insertion in this pair of channels is enabled; false: The deadtime insertion in this pair of channels is disabled.

struct __ftm__chnl__pwm__config__param

#include <fsl_ftm.h> Options to configure a FTM channel using precise setting.

Public Members

ftm_chnl_t chnlNumber

The channel/channel pair number. In combined mode, this represents the channel pair number.

ftm_pwm_level_select_t level

PWM output active level select.

uint16_t dutyValue

PWM pulse width, the uint of this value is timer ticks.

uint16_t firstEdgeValue

Used only in kFTM_AsymmetricalCombinedPwm mode to generate an asymmetrical PWM. Specifies the delay to the first edge in a PWM period. If unsure leave as 0, uint of this value is timer ticks.

bool enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

bool enableDeadtime

Used only in combined PWM mode with enable complementary. true: The deadtime insertion in this pair of channels is enabled; false: The deadtime insertion in this pair of channels is disabled.

struct _ftm_chnl_param

#include <fsl_ftm.h> General options to configure a FTM channel using precise setting.

Public Members

ftm_pwm_mode_t mode

PWM output mode.

ftm_pwm_level_select_t level

PWM output active level select.

uint16_t initialValue

FTM counter initial value.

uint16_t moduloValue

FTM counter modulo value.

uint16_t chnlValue

FTM channel n match value.

uint16_t combinedChnlValue

FTM combined channel n+1 match value, used only in (modified) combined PWM mode.

bool enableComplementary

Used only in combined PWM mode. true: The combined channels output complementary signals; false: The combined channels output same signals;

bool enableDeadtime

Used only in combined PWM mode with enable complementary. true: The deadtime insertion in this pair of channels is enabled; false: The deadtime insertion in this pair of channels is disabled.

struct _ftm_dual_edge_capture_param

#include <fsl_ftm.h> FlexTimer dual edge capture parameters.

Public Members

ftm_dual_edge_capture_mode_t mode

Dual Edge Capture mode

ftm_input_capture_edge_t currChanEdgeMode

Input capture edge select for channel n

ftm_input_capture_edge_t nextChanEdgeMode

Input capture edge select for channel n+1

struct _ftm_phase_param

#include <fsl_ftm.h> FlexTimer quadrature decode phase parameters.

Public Members

bool enablePhaseFilter

True: enable phase filter; false: disable filter

uint32_t phaseFilterVal

Filter value, used only if phase filter is enabled

ftm_phase_polarity_t phasePolarity

Phase polarity

struct _ftm_fault_param

#include <fsl_ftm.h> Structure is used to hold the parameters to configure a FTM fault.

Public Members

bool enableFaultInput

True: Fault input is enabled; false: Fault input is disabled

bool faultLevel

True: Fault polarity is active low; in other words, '0' indicates a fault; False: Fault polarity is active high

bool useFaultFilter

True: Use the filtered fault signal; False: Use the direct path from fault input

struct _ftm_config

#include <fsl_ftm.h> FTM configuration structure.

This structure holds the configuration settings for the FTM peripheral. To initialize this structure to reasonable defaults, call the *FTM_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

ftm_clock_prescale_t prescale

FTM clock prescale value

ftm_bdm_mode_t bdmMode

FTM behavior in BDM mode

uint32_t pwmSyncMode

Synchronization methods to use to update buffered registers; Multiple update modes can be used by providing an OR'ed list of options available in enumeration *ftm_pwm_sync_method_t*.

uint32_t reloadPoints

FTM reload points; When using this, the PWM synchronization is not required. Multiple reload points can be used by providing an OR'ed list of options available in enumeration *ftm_reload_point_t*.

ftm_fault_mode_t faultMode

FTM fault control mode

uint8_t faultFilterValue

Fault input filter value

ftm_deadtime_prescale_t deadTimePrescale

The dead time prescalar value

uint32_t deadTimeValue

The dead time value deadTimeValue's available range is 0-1023 when register has DTVALEX, otherwise its available range is 0-63.

uint32_t extTriggers

External triggers to enable. Multiple trigger sources can be enabled by providing an OR'ed list of options available in enumeration *ftm_external_trigger_t*.

uint8_t chnlInitState

Defines the initialization value of the channels in OUTINT register

uint8_t chnlPolarity

Defines the output polarity of the channels in POL register

bool useGlobalTimeBase

True: Use of an external global time base is enabled; False: disabled

bool swTriggerResetCount

FTM counter synchronization activated by software trigger, avtive when (syncMethod & FTM_SYNC_SWSYNC_MASK) != 0U

bool hwTriggerResetCount

FTM counter synchronization activated by hardware trigger, avtive when (syncMethod & (FTM_SYNC_TRIG0_MASK | FTM_SYNC_TRIG1_MASK | FTM_SYNC_TRIG2_MASK)) != 0U

2.23 GPIO: General-Purpose Input/Output Driver

FSL_GPIO_DRIVER_VERSION
GPIO driver version.

enum _gpio_pin_direction
GPIO direction definition.

Values:

enumerator kGPIO_DigitalInput
Set current pin as digital input

enumerator kGPIO_DigitalOutput
Set current pin as digital output

enum _gpio_checker_attribute
GPIO checker attribute.

Values:

enumerator kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureRW
User nonsecure:Read+Write; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW
User nonsecure:Read; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW
User nonsecure:None; User Secure:Read+Write; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW
User nonsecure:Read; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW
User nonsecure:None; User Secure:Read; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW
User nonsecure:None; User Secure:None; Privileged Secure:Read+Write

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR
User nonsecure:None; User Secure:None; Privileged Secure:Read

enumerator kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN
User nonsecure:None; User Secure:None; Privileged Secure:None

enumerator kGPIO_IgnoreAttributeCheck
Ignores the attribute check

typedef enum *_gpio_pin_direction* gpio_pin_direction_t
GPIO direction definition.

typedef enum *_gpio_checker_attribute* gpio_checker_attribute_t

GPIO checker attribute.

typedef struct *_gpio_pin_config* gpio_pin_config_t

The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the *PORT_SetPinConfig()*.

GPIO_FIT_REG(value)

struct _gpio_pin_config

#include <fsl_gpio.h> The GPIO pin configuration structure.

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the *PORT_SetPinConfig()*.

Public Members

gpio_pin_direction_t pinDirection

GPIO direction, input or output

uint8_t outputLogic

Set a default output logic, which has no use in input

2.24 GPIO Driver

void GPIO_PortInit(GPIO_Type *base)

Initializes the GPIO peripheral.

This function ungates the GPIO clock.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PortDenit(GPIO_Type *base)

Denitializes the GPIO peripheral.

Parameters

- base – GPIO peripheral base pointer.

void GPIO_PinInit(GPIO_Type *base, uint32_t pin, const *gpio_pin_config_t* *config)

Initializes a GPIO pin used by the board.

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the *GPIO_PinInit()* function.

This is an example to define an input pin or an output pin configuration.

```
Define a digital input pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalInput,
  0,
}
Define a digital output pin configuration,
gpio_pin_config_t config =
{
  kGPIO_DigitalOutput,
  0,
}
```

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO port pin number
- config – GPIO pin configuration pointer

static inline void GPIO_PinWrite(GPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the multiple GPIO pins to the logic 1 or 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- pin – GPIO pin number
- output – GPIO pin output logic level.
  - 0: corresponding pin output low-logic level.
  - 1: corresponding pin output high-logic level.

static inline void GPIO_PortSet(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 1.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortClear(GPIO_Type *base, uint32_t mask)

Sets the output level of the multiple GPIO pins to the logic 0.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline void GPIO_PortToggle(GPIO_Type *base, uint32_t mask)

Reverses the current output logic of the multiple GPIO pins.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

static inline uint32_t GPIO_PinRead(GPIO_Type *base, uint32_t pin)

Reads the current input value of the GPIO port.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)

- pin – GPIO pin number

Return values

GPIO – port input value

- 0: corresponding pin input low-logic level.
- 1: corresponding pin input high-logic level.

uint32_t GPIO_PortGetInterruptFlags(GPIO_Type *base)

Reads the GPIO port interrupt status flag.

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)

Return values

The – current GPIO port interrupt status flag, for example, 0x00010001 means the pin 0 and 17 have the interrupt.

void GPIO_PortClearInterruptFlags(GPIO_Type *base, uint32_t mask)

Clears multiple GPIO pin interrupt status flags.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- mask – GPIO pin number macro

void GPIO_CheckAttributeBytes(GPIO_Type *base, *gpio_checker_attribute_t* attribute)

brief The GPIO module supports a device-specific number of data ports, organized as 32-bit words/8-bit Bytes. Each 32-bit/8-bit data port includes a GACR register, which defines the byte-level attributes required for a successful access to the GPIO programming model. If the GPIO module's GACR register organized as 32-bit words, the attribute controls for the 4 data bytes in the GACR follow a standard little endian data convention.

Parameters

- base – GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
- attribute – GPIO checker attribute

2.25 I2C: Inter-Integrated Circuit Driver

2.26 I2C Driver

void I2C_MasterInit(I2C_Type *base, const *i2c_master_config_t* *masterConfig, uint32_t srcClock_Hz)

Initializes the I2C peripheral. Call this API to ungate the I2C clock and configure the I2C with master configuration.

Note: This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can be custom filled or it can be set with default values by using the *I2C_MasterGetDefaultConfig()*. After calling this API, the master is ready to transfer. This is an example.

```
i2c_master_config_t config = {
.enableMaster = true,
.enableStopHold = false,
.highDrive = false,
.baudRate_Bps = 100000,
.glitchFilterWidth = 0
};
I2C_MasterInit(I2C0, &config, 12000000U);
```

Parameters

- base – I2C base pointer
- masterConfig – A pointer to the master configuration structure
- srcClock_Hz – I2C peripheral clock frequency in Hz

void I2C_SlaveInit(I2C_Type *base, const *i2c_slave_config_t* *slaveConfig, uint32_t srcClock_Hz)

Initializes the I2C peripheral. Call this API to ungate the I2C clock and initialize the I2C with the slave configuration.

Note: This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can partly be set with default values by *I2C_SlaveGetDefaultConfig()* or it can be custom filled by the user. This is an example.

```
i2c_slave_config_t config = {
.enableSlave = true,
.enableGeneralCall = false,
.addressingMode = kI2C_Address7bit,
.slaveAddress = 0x1DU,
.enableWakeUp = false,
.enablehighDrive = false,
.enableBaudRateCtl = false,
.sclStopHoldTime_ns = 4000
};
I2C_SlaveInit(I2C0, &config, 12000000U);
```

Parameters

- base – I2C base pointer
- slaveConfig – A pointer to the slave configuration structure
- srcClock_Hz – I2C peripheral clock frequency in Hz

void I2C_MasterDeinit(I2C_Type *base)

De-initializes the I2C master peripheral. Call this API to gate the I2C clock. The I2C master module can't work unless the I2C_MasterInit is called.

Parameters

- base – I2C base pointer

void I2C_SlaveDeinit(I2C_Type *base)

De-initializes the I2C slave peripheral. Calling this API gates the I2C clock. The I2C slave module can't work unless the I2C_SlaveInit is called to enable the clock.

Parameters

- base – I2C base pointer

uint32_t I2C_GetInstance(I2C_Type *base)

Get instance number for I2C module.

Parameters

- base – I2C peripheral base address.

void I2C_MasterGetDefaultConfig(*i2c_master_config_t* *masterConfig)

Sets the I2C master configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in the I2C_MasterConfigure(). Use the initialized structure unchanged in the I2C_MasterConfigure() or modify the structure before calling the I2C_MasterConfigure(). This is an example.

```
i2c_master_config_t config;
I2C_MasterGetDefaultConfig(&config);
```

Parameters

- masterConfig – A pointer to the master configuration structure.

void I2C_SlaveGetDefaultConfig(*i2c_slave_config_t* *slaveConfig)

Sets the I2C slave configuration structure to default values.

The purpose of this API is to get the configuration structure initialized for use in the I2C_SlaveConfigure(). Modify fields of the structure before calling the I2C_SlaveConfigure(). This is an example.

```
i2c_slave_config_t config;
I2C_SlaveGetDefaultConfig(&config);
```

Parameters

- slaveConfig – A pointer to the slave configuration structure.

static inline void I2C_Enable(I2C_Type *base, bool enable)

Enables or disables the I2C peripheral operation.

Parameters

- base – I2C base pointer
- enable – Pass true to enable and false to disable the module.

uint32_t I2C_MasterGetStatusFlags(I2C_Type *base)

Gets the I2C status flags.

Parameters

- base – I2C base pointer

Returns

status flag, use status flag to AND *_i2c_flags* to get the related status.

static inline uint32_t I2C_SlaveGetStatusFlags(I2C_Type *base)

Gets the I2C status flags.

Parameters

- base – I2C base pointer

Returns

status flag, use status flag to AND *_i2c_flags* to get the related status.

static inline void I2C_MasterClearStatusFlags(I2C_Type *base, uint32_t statusMask)

Clears the I2C status flag state.

The following status register flags can be cleared kI2C_ArbitrationLostFlag and kI2C_IntPendingFlag.

Parameters

- base – I2C base pointer
- statusMask – The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values:
  - kI2C_StartDetectFlag (if available)
  - kI2C_StopDetectFlag (if available)
  - kI2C_ArbitrationLostFlag
  - kI2C_IntPendingFlagFlag

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 kI2C_ArbitrationLostFlag and kI2C_IntPendingFlag

Parameters

- base – I2C base pointer
- statusMask – The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values:
  - kI2C_StartDetectFlag (if available)
  - kI2C_StopDetectFlag (if available)
  - kI2C_ArbitrationLostFlag
  - kI2C_IntPendingFlagFlag

void I2C_EnableInterrupts(I2C_Type *base, uint32_t mask)

Enables I2C interrupt requests.

Parameters

- base – I2C base pointer
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kI2C_GlobalInterruptEnable
  - kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable
  - kI2C_SdaTimeoutInterruptEnable

void I2C_DisableInterrupts(I2C_Type *base, uint32_t mask)

Disables I2C interrupt requests.

Parameters

- base – I2C base pointer
- mask – interrupt source The parameter can be combination of the following source if defined:
  - kI2C_GlobalInterruptEnable
  - kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable
  - kI2C_SdaTimeoutInterruptEnable

static inline void I2C_EnableDMA(I2C_Type *base, bool enable)

Enables/disables the I2C DMA interrupt.

Parameters

- base – I2C base pointer
- enable – true to enable, false to disable

static inline uint32_t I2C_GetDataRegAddr(I2C_Type *base)

Gets the I2C tx/rx data register address. This API is used to provide a transfer address for I2C DMA transfer configuration.

Parameters

- base – I2C base pointer

Returns

data register address

void I2C_MasterSetBaudRate(I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the I2C master transfer baud rate.

Parameters

- base – I2C base pointer
- baudRate_Bps – the baud rate value in bps
- srcClock_Hz – Source clock

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.

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 uint8_t *txBuff, size_t txSize, uint32_t flags)

Performs a polling send transaction on the I2C bus.

Parameters

- base – The I2C peripheral base pointer.
- 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 decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

Return values

- kStatus_Success – Successfully complete the data transmission.
- kStatus_I2C_ArbitrationLost – Transfer error, arbitration lost.
- kStataus_I2C_Nak – Transfer error, receive NAK during transfer.

status_t I2C_MasterReadBlocking(I2C_Type *base, uint8_t *rxBuff, size_t rxSize, uint32_t flags)

Performs a polling receive transaction on the I2C bus.

Note: The I2C_MasterReadBlocking function stops the bus before reading the final byte. Without stopping the bus prior for the final read, the bus issues another read, resulting in garbage data being read into the data register.

Parameters

- base – I2C peripheral base pointer.
- rxBuff – The pointer to the data to store the received data.
- rxSize – The length in bytes of the data to be received.
- flags – Transfer control flag to decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

Return values

- kStatus_Success – Successfully complete the data transmission.
- kStatus_I2C_Timeout – Send stop signal failed, timeout.

status_t I2C_SlaveWriteBlocking(I2C_Type *base, const uint8_t *txBuff, size_t txSize)

Performs a polling send transaction on the I2C bus.

Parameters

- base – The I2C peripheral base pointer.
- txBuff – The pointer to the data to be transferred.
- txSize – The length in bytes of the data to be transferred.

Return values

- kStatus_Success – Successfully complete the data transmission.

- kStatus_I2C_ArbitrationLost – Transfer error, arbitration lost.
- kStataus_I2C_Nak – Transfer error, receive NAK during transfer.

status_t I2C_SlaveReadBlocking(I2C_Type *base, uint8_t *rxBuff, size_t rxSize)

Performs a polling receive transaction on the I2C bus.

Parameters

- base – I2C peripheral base pointer.
- rxBuff – The pointer to the data to store the received data.
- rxSize – The length in bytes of the data to be received.

Return values

- kStatus_Success – Successfully complete data receive.
- kStatus_I2C_Timeout – Wait status flag timeout.

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.

void I2C_MasterTransferCreateHandle(I2C_Type *base, *i2c_master_handle_t* *handle, *i2c_master_transfer_callback_t* callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_master_handle_t structure to store the transfer state.
- callback – pointer to user callback function.
- userData – user parameter passed to the callback function.

status_t I2C_MasterTransferNonBlocking(I2C_Type *base, *i2c_master_handle_t* *handle, *i2c_master_transfer_t* *xfer)

Performs a master interrupt non-blocking transfer on the I2C bus.

Note: Calling the API returns immediately after transfer initiates. The user needs to call I2C_MasterGetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_I2C_Busy, the transfer is finished.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_master_handle_t structure which stores the transfer state.
- xfer – pointer to i2c_master_transfer_t structure.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_I2C_Busy – Previous transmission still not finished.
- kStatus_I2C_Timeout – Transfer error, wait signal timeout.

status_t I2C_MasterTransferGetCount(I2C_Type *base, *i2c_master_handle_t* *handle, size_t *count)

Gets the master transfer status during a interrupt non-blocking transfer.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_master_handle_t structure which stores the transfer state.
- count – Number of bytes transferred so far by the non-blocking transaction.

Return values

- kStatus_InvalidArgument – count is Invalid.
- kStatus_Success – Successfully return the count.

status_t I2C_MasterTransferAbort(I2C_Type *base, *i2c_master_handle_t* *handle)

Aborts an interrupt non-blocking transfer early.

Note: This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_master_handle_t structure which stores the transfer state

Return values

- kStatus_I2C_Timeout – Timeout during polling flag.
- kStatus_Success – Successfully abort the transfer.

void I2C_MasterTransferHandleIRQ(I2C_Type *base, void *i2cHandle)

Master interrupt handler.

Parameters

- base – I2C base pointer.

- i2cHandle – pointer to i2c_master_handle_t structure.

void I2C_SlaveTransferCreateHandle(I2C_Type *base, *i2c_slave_handle_t* *handle, *i2c_slave_transfer_callback_t* callback, void *userData)

Initializes the I2C handle which is used in transactional functions.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_slave_handle_t structure to store the transfer state.
- callback – pointer to user callback function.
- userData – user parameter passed to the callback function.

status_t I2C_SlaveTransferNonBlocking(I2C_Type *base, *i2c_slave_handle_t* *handle, uint32_t eventMask)

Starts accepting slave transfers.

Call this API after calling the *I2C_SlaveInit()* and *I2C_SlaveTransferCreateHandle()* to start processing transactions driven by an I2C master. The slave monitors the I2C bus and passes events to the callback that was passed into the call to *I2C_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 *i2c_slave_transfer_event_t* enumerators for the events you wish to receive. The kI2C_SlaveTransmitEvent and kLPI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, 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.

void I2C_SlaveTransferAbort(I2C_Type *base, *i2c_slave_handle_t* *handle)

Aborts the slave transfer.

Note: This API can be called at any time to stop slave for handling the bus events.

Parameters

- base – I2C base pointer.
- handle – pointer to i2c_slave_handle_t structure which stores the transfer state.

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, void *i2cHandle)

Slave interrupt handler.

Parameters

- base – I2C base pointer.
- i2cHandle – pointer to i2c_slave_handle_t structure which stores the transfer state

void I2C_DriverIRQHandler(uint32_t instance)

I2C driver IRQ handler common entry.

Parameters

- instance – I2C peripheral instance number.

FSL_I2C_DRIVER_VERSION

I2C driver version.

I2C status return codes.

Values:

enumerator kStatus_I2C_Busy

I2C is busy with current transfer.

enumerator kStatus_I2C_Idle

Bus is Idle.

enumerator kStatus_I2C_Nak

NAK received during transfer.

enumerator kStatus_I2C_ArbitrationLost

Arbitration lost during transfer.

enumerator kStatus_I2C_Timeout

Timeout polling status flags.

enumerator kStatus_I2C_Addr_Nak

NAK received during the address probe.

enum _i2c_flags

I2C peripheral flags.

Note: These enumerations are meant to be OR'd together to form a bit mask.

Values:

enumerator kI2C_ReceiveNakFlag

I2C receive NAK flag.

enumerator kI2C_IntPendingFlag

I2C interrupt pending flag. This flag can be cleared.

enumerator kI2C_TransferDirectionFlag

I2C transfer direction flag.

enumerator kI2C_RangeAddressMatchFlag

I2C range address match flag.

enumerator kI2C_ArbitrationLostFlag

I2C arbitration lost flag. This flag can be cleared.

enumerator kI2C_BusBusyFlag

I2C bus busy flag.

enumerator kI2C_AddressMatchFlag

I2C address match flag.

enumerator kI2C_TransferCompleteFlag

I2C transfer complete flag.

enumerator kI2C_StopDetectFlag

I2C stop detect flag. This flag can be cleared.

enumerator kI2C_StartDetectFlag

I2C start detect flag. This flag can be cleared.

enum _i2c_interrupt_enable

I2C feature interrupt source.

Values:

enumerator kI2C_GlobalInterruptEnable

I2C global interrupt.

enumerator kI2C_StopDetectInterruptEnable

I2C stop detect interrupt.

enumerator kI2C_StartStopDetectInterruptEnable

I2C start&stop detect interrupt.

enum _i2c_direction

The direction of master and slave transfers.

Values:

enumerator kI2C_Write

Master transmits to the slave.

enumerator kI2C_Read

Master receives from the slave.

enum _i2c_slave_address_mode

Addressing mode.

Values:

enumerator kI2C_Address7bit

7-bit addressing mode.

enumerator kI2C_RangeMatch

Range address match addressing mode.

enum _i2c_master_transfer_flags

I2C transfer control flag.

Values:

enumerator kI2C_TransferDefaultFlag

A transfer starts with a start signal, stops with a stop signal.

enumerator kI2C_TransferNoStartFlag

A transfer starts without a start signal, only support write only or write+read with no start flag, do not support read only with no start flag.

enumerator kI2C_TransferRepeatedStartFlag

A transfer starts with a repeated start signal.

enumerator kI2C_TransferNoStopFlag

A transfer ends without a stop signal.

enum _i2c_slave_transfer_event

Set of events sent to the callback for nonblocking 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()* 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
A callback is requested to provide data to transmit (slave-transmitter role).

enumerator kI2C_SlaveReceiveEvent
A callback is requested to provide a buffer in which to place received data (slave-receiver role).

enumerator kI2C_SlaveTransmitAckEvent
A callback needs to either transmit an ACK or NACK.

enumerator kI2C_SlaveStartEvent
A start/repeated start was detected.

enumerator kI2C_SlaveCompletionEvent
A stop was detected or finished transfer, completing the transfer.

enumerator kI2C_SlaveGenaralcallEvent
Received the general call address after a start or repeated start.

enumerator kI2C_SlaveAllEvents
A bit mask of all available events.

Common sets of flags used by the driver.

Values:

enumerator kClearFlags
All flags which are cleared by the driver upon starting a transfer.

enumerator kIrqFlags

typedef enum *_i2c_direction* i2c_direction_t
The direction of master and slave transfers.

typedef enum *_i2c_slave_address_mode* i2c_slave_address_mode_t
Addressing mode.

typedef enum *_i2c_slave_transfer_event* i2c_slave_transfer_event_t
Set of events sent to the callback for nonblocking 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()* 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_master_config* i2c_master_config_t

I2C master user configuration.

typedef struct *_i2c_slave_config* i2c_slave_config_t

I2C slave user configuration.

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* status, void *userData)

I2C master transfer callback typedef.

typedef struct *_i2c_slave_handle* i2c_slave_handle_t

I2C slave handle typedef.

typedef struct *_i2c_master_transfer* i2c_master_transfer_t

I2C master transfer structure.

typedef struct *_i2c_slave_transfer* i2c_slave_transfer_t

I2C slave transfer structure.

typedef void (*i2c_slave_transfer_callback_t)(I2C_Type *base, *i2c_slave_transfer_t* *xfer, void *userData)

I2C slave transfer callback typedef.

I2C_RETRY_TIMES

Retry times for waiting flag.

I2C_MASTER_FACK_CONTROL

Mater Fast ack control, control if master needs to manually write ack, this is used to low the speed of transfer for SoCs with feature FSL_FEATURE_I2C_HAS_DOUBLE_BUFFERING.

I2C_HAS_STOP_DETECT

struct i2c_master_config

#include <fsl_i2c.h> I2C master user configuration.

Public Members

bool enableMaster

Enables the I2C peripheral at initialization time.

bool enableStopHold

Controls the stop hold enable.

bool enableDoubleBuffering

Controls double buffer enable; notice that enabling the double buffer disables the clock stretch.

uint32_t baudRate_Bps

Baud rate configuration of I2C peripheral.

uint8_t glitchFilterWidth

Controls the width of the glitch.

struct _i2c_slave_config

#include <fsl_i2c.h> I2C slave user configuration.

Public Members

bool enableSlave

Enables the I2C peripheral at initialization time.

bool enableGeneralCall

Enables the general call addressing mode.

bool enableWakeUp

Enables/disables waking up MCU from low-power mode.

bool enableDoubleBuffering

Controls a double buffer enable; notice that enabling the double buffer disables the clock stretch.

bool enableBaudRateCtl

Enables/disables independent slave baud rate on SCL in very fast I2C modes.

uint16_t slaveAddress

A slave address configuration.

uint16_t upperAddress

A maximum boundary slave address used in a range matching mode.

i2c_slave_address_mode_t addressingMode

An addressing mode configuration of i2c_slave_address_mode_config_t.

uint32_t sclStopHoldTime_ns

the delay from the rising edge of SCL (I2C clock) to the rising edge of SDA (I2C data) while SCL is high (stop condition), SDA hold time and SCL start hold time are also configured according to the SCL stop hold time.

struct _i2c_master_transfer

#include <fsl_i2c.h> I2C master transfer structure.

Public Members

uint32_t flags
A transfer flag which controls the transfer.

uint8_t slaveAddress
7-bit slave address.

i2c_direction_t direction
A transfer direction, read or write.

uint32_t subaddress
A sub address. Transferred MSB first.

uint8_t subaddressSize
A size of the command buffer.

uint8_t *volatile data
A transfer buffer.

volatile size_t dataSize
A transfer size.

struct __i2c_master_handle
#include <fsl_i2c.h> I2C master handle structure.

Public Members

i2c_master_transfer_t transfer
I2C master transfer copy.

size_t transferSize
Total bytes to be transferred.

uint8_t state
A transfer state maintained during transfer.

i2c_master_transfer_callback_t completionCallback
A callback function called when the transfer is finished.

void *userData
A callback parameter passed to the callback function.

struct __i2c_slave_transfer
#include <fsl_i2c.h> I2C slave transfer structure.

Public Members

i2c_slave_transfer_event_t event
A reason that the callback is invoked.

uint8_t *volatile data
A transfer buffer.

volatile size_t dataSize
A transfer size.

status_t completionStatus
Success or error code describing how the transfer completed. Only applies for *kI2C_SlaveCompletionEvent*.

size_t transferredCount
A number of bytes actually transferred since the start or since the last repeated start.

struct __i2c_slave_handle
#include <fsl_i2c.h> I2C slave handle structure.

Public Members

volatile bool isBusy
Indicates whether a transfer is busy.

i2c_slave_transfer_t transfer
I2C slave transfer copy.

uint32_t eventMask
A mask of enabled events.

i2c_slave_transfer_callback_t callback
A callback function called at the transfer event.

void *userData
A callback parameter passed to the callback.

2.27 I2C eDMA Driver

void I2C_MasterCreateEDMAHandle(I2C_Type *base, *i2c_master_edma_handle_t* *handle, *i2c_master_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *edmaHandle)
Initializes the I2C handle which is used in transactional functions.

Parameters

- base – I2C peripheral base address.

- handle – A pointer to the i2c_master_edma_handle_t structure.
- callback – A pointer to the user callback function.
- userData – A user parameter passed to the callback function.
- edmaHandle – eDMA handle pointer.

status_t I2C_MasterTransferEDMA(I2C_Type *base, *i2c_master_edma_handle_t* *handle, *i2c_master_transfer_t* *xfer)

Performs a master eDMA non-blocking transfer on the I2C bus.

Parameters

- base – I2C peripheral base address.
- handle – A pointer to the i2c_master_edma_handle_t structure.
- xfer – A pointer to the transfer structure of i2c_master_transfer_t.

Return values

- kStatus_Success – Successfully completed the data transmission.
- kStatus_I2C_Busy – A previous transmission is still not finished.
- kStatus_I2C_Timeout – Transfer error, waits for a signal timeout.
- kStatus_I2C_ArbitrationLost – Transfer error, arbitration lost.
- kStataus_I2C_Nak – Transfer error, receive NAK during transfer.

status_t I2C_MasterTransferGetCountEDMA(I2C_Type *base, *i2c_master_edma_handle_t* *handle, size_t *count)

Gets a master transfer status during the eDMA non-blocking transfer.

Parameters

- base – I2C peripheral base address.
- handle – A pointer to the i2c_master_edma_handle_t structure.
- count – A number of bytes transferred by the non-blocking transaction.

void I2C_MasterTransferAbortEDMA(I2C_Type *base, *i2c_master_edma_handle_t* *handle)

Aborts a master eDMA non-blocking transfer early.

Parameters

- base – I2C peripheral base address.
- handle – A pointer to the i2c_master_edma_handle_t structure.

FSL_I2C_EDMA_DRIVER_VERSION

I2C EDMA driver version.

typedef struct *_i2c_master_edma_handle* i2c_master_edma_handle_t

Retry times for waiting flag.

I2C master eDMA handle typedef.

typedef void (*i2c_master_edma_transfer_callback_t)(I2C_Type *base, *i2c_master_edma_handle_t* *handle, *status_t* status, void *userData)

I2C master eDMA transfer callback typedef.

struct _i2c_master_edma_handle

#include <fsl_i2c_edma.h> I2C master eDMA transfer structure.

Public Members

i2c_master_transfer_t transfer
I2C master transfer structure.

size_t transferSize
Total bytes to be transferred.

uint8_t nbytes
eDMA minor byte transfer count initially configured.

uint8_t state
I2C master transfer status.

edma_handle_t *dmaHandle
The eDMA handler used.

i2c_master_edma_transfer_callback_t completionCallback
A callback function called after the eDMA transfer is finished.

void *userData
A callback parameter passed to the callback function.

2.28 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

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.

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_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(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_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.

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_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.

MCUX_DEPRECATED
Deprecated APIs.

MCUX_DEPRECATED_MACRO
Deprecated macros.

MCUX_EXPERIMENTAL
Experimental APIs.

MCUX_EXPERIMENTAL_MACRO
Experimental macros.

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.

Enable 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 enables 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 number.

Return values

- kStatus_Success – Interrupt enabled successfully
- kStatus_Fail – Failed to enable the interrupt

static inline *status_t* DisableIRQ(IRQn_Type interrupt)

Disable specific interrupt.

Disable 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 disables 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 number.

Return values

- kStatus_Success – Interrupt disabled successfully
- kStatus_Fail – Failed to disable the interrupt

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 uint32_t DisableGlobalIRQ(void)

Disable the global IRQ.

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the *EnableGlobalIRQ()*.

Returns

Current primask value.

static inline void EnableGlobalIRQ(uint32_t primask)

Enable the global IRQ.

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management

mechanism of primask. User is required to use the *EnableGlobalIRQ()* and *DisableGlobalIRQ()* in pair.

Parameters

- primask – value of primask register to be restored. The primask value is supposed to be provided by the *DisableGlobalIRQ()*.

static inline bool _SDK_AtomicLocalCompareAndSet(uint32_t *addr, uint32_t expected, uint32_t newValue)

static inline uint32_t _SDK_AtomicTestAndSet(uint32_t *addr, uint32_t newValue)

FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ

Macro to use the default weak IRQ handler in drivers.

MAKE_STATUS(group, code)

Construct a status code value from a group and code number.

MAKE_VERSION(major, minor, bugfix)

Construct the version number for drivers.

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

```
| Unused    || Major Version || Minor Version ||  Bug Fix    |
31       25  24          17  16          9  8           0
```

ARRAY_SIZE(x)

Computes the number of elements in an array.

SUPPRESS_FALL_THROUGH_WARNING()

For switch case code block, if case section ends without "break;" statement, there wil be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc. To suppress this warning, "SUPPRESS_FALL_THROUGH_WARNING();" need to be added at the end of each case section which misses "break;"statement.

MSDK_REG_SECURE_ADDR(x)

Convert the register address to the one used in secure mode.

MSDK_REG_NONSECURE_ADDR(x)

Convert the register address to the one used in non-secure mode.

MSDK_HAS_DWT_CYCCNT

The chip supports DWT CYCCNT or not.

MSDK_INVALID_IRQ_HANDLER

Invalid IRQ handler address.

2.29 Lin_lpuart_driver

FSL_LIN_LPUART_DRIVER_VERSION

LIN LPUART driver version.

enum _lin_lpuart_stop_bit_count

Values:

enumerator kLPUART_OneStopBit
One stop bit

enumerator kLPUART_TwoStopBit
Two stop bits

enum _lin_lpuart_flags
Values:

enumerator kLPUART_TxDataRegEmptyFlag
Transmit data register empty flag, sets when transmit buffer is empty

enumerator kLPUART_TransmissionCompleteFlag
Transmission complete flag, sets when transmission activity complete

enumerator kLPUART_RxDataRegFullFlag
Receive data register full flag, sets when the receive data buffer is full

enumerator kLPUART_IdleLineFlag
Idle line detect flag, sets when idle line detected

enumerator kLPUART_RxOverrunFlag
Receive Overrun, sets when new data is received before data is read from receive register

enumerator kLPUART_NoiseErrorFlag
Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets

enumerator kLPUART_FramingErrorFlag
Frame error flag, sets if logic 0 was detected where stop bit expected

enumerator kLPUART_ParityErrorFlag
If parity enabled, sets upon parity error detection

enumerator kLPUART_LinBreakFlag
LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled

enumerator kLPUART_RxActiveEdgeFlag
Receive pin active edge interrupt flag, sets when active edge detected

enumerator kLPUART_RxActiveFlag
Receiver Active Flag (RAF), sets at beginning of valid start bit

enumerator kLPUART_DataMatch1Flag
The next character to be read from LPUART_DATA matches MA1

enumerator kLPUART_DataMatch2Flag
The next character to be read from LPUART_DATA matches MA2

enumerator kLPUART_NoiseErrorInRxDataRegFlag
NOISY bit, sets if noise detected in current data word

enumerator kLPUART_ParityErrorInRxDataRegFlag
PARITY bit, sets if noise detected in current data word

enumerator kLPUART_TxFifoEmptyFlag
TXEMPT bit, sets if transmit buffer is empty

enumerator kLPUART_RxFifoEmptyFlag
RXEMPT bit, sets if receive buffer is empty

enumerator kLPUART_TxFifoOverflowFlag
TXOF bit, sets if transmit buffer overflow occurred

enumerator kLPUART_RxFifoUnderflowFlag
RXUF bit, sets if receive buffer underflow occurred

enum _lin_lpuart_interrupt_enable
Values:

enumerator kLPUART_LinBreakInterruptEnable
LIN break detect.

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge.

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty.

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete.

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full.

enumerator kLPUART_IdleLineInterruptEnable
Idle line.

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun.

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag.

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag.

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag.

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow.

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow.

enum _lin_lpuart_status
Values:

enumerator kStatus_LPUART_TxBusy
TX busy

enumerator kStatus_LPUART_RxBusy
RX busy

enumerator kStatus_LPUART_TxIdle
LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle
LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge
TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge
RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually
Some flag can't manually clear

enumerator kStatus_LPUART_Error
Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun
LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun
LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError
LPUART noise error.

enumerator kStatus_LPUART_FramingError
LPUART framing error.

enumerator kStatus_LPUART_ParityError
LPUART parity error.

enum lin_lpuart_bit_count_per_char_t
Values:

enumerator LPUART_8_BITS_PER_CHAR
8-bit data characters

enumerator LPUART_9_BITS_PER_CHAR
9-bit data characters

enumerator LPUART_10_BITS_PER_CHAR
10-bit data characters

typedef enum *_lin_lpuart_stop_bit_count* lin_lpuart_stop_bit_count_t

static inline bool LIN_LPUART_GetRxDataPolarity(const LPUART_Type *base)

static inline void LIN_LPUART_SetRxDataPolarity(LPUART_Type *base, bool polarity)

static inline void LIN_LPUART_WriteByte(LPUART_Type *base, uint8_t data)

static inline void LIN_LPUART_ReadByte(const LPUART_Type *base, uint8_t *readData)

status_t LIN_LPUART_CalculateBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *osr, uint16_t *sbr)
Calculates the best osr and sbr value for configured baudrate.

Parameters

- base – LPUART peripheral base address
- baudRate_Bps – user configuration structure of type #lin_user_config_t
- srcClock_Hz – pointer to the LIN_LPUART driver state structure
- osr – pointer to osr value
- sbr – pointer to sbr value

Returns
An error code or lin_status_t

void LIN_LPUART_SetBaudRate(LPUART_Type *base, uint32_t *osr, uint16_t *sbr)
Configure baudrate according to osr and sbr value.

Parameters

- base – LPUART peripheral base address
- osr – pointer to osr value
- sbr – pointer to sbr value

lin_status_t LIN_LPUART_Init(LPUART_Type *base, lin_user_config_t *linUserConfig, lin_state_t *linCurrentState, uint32_t linSourceClockFreq)

Initializes an LIN_LPUART instance for LIN Network.

The caller provides memory for the driver state structures during initialization. The user must select the LIN_LPUART clock source in the application to initialize the LIN_LPUART. This function initializes a LPUART instance for operation. This function will initialize the run-time state structure to keep track of the on-going transfers, initialize the module to user defined settings and default settings, set break field length to be 13 bit times minimum, enable the break detect interrupt, Rx complete interrupt, frame error detect interrupt, and enable the LPUART module transmitter and receiver

Parameters

- base – LPUART peripheral base address
- linUserConfig – user configuration structure of type #lin_user_config_t
- linCurrentState – pointer to the LIN_LPUART driver state structure
- linSourceClockFreq – LIN source clock frequency in Hz

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_Deinit(LPUART_Type *base)

Shuts down the LIN_LPUART by disabling interrupts and transmitter/receiver.

Parameters

- base – LPUART peripheral base address

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_SendFrameDataBlocking(LPUART_Type *base, const uint8_t *txBuff, uint8_t txSize, uint32_t timeoutMSec)

Sends Frame data out through the LIN_LPUART module using blocking method. This function will calculate the checksum byte and send it with the frame data. Blocking means that the function does not return until the transmission is complete.

Parameters

- base – LPUART peripheral base address
- txBuff – source buffer containing 8-bit data chars to send
- txSize – the number of bytes to send
- timeoutMSec – timeout value in milli seconds

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_SendFrameData(LPUART_Type *base, const uint8_t *txBuff, uint8_t txSize)

Sends frame data out through the LIN_LPUART module using non-blocking method. This enables an a-sync method for transmitting data. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete. This function will calculate the checksum byte and send it with the frame data.

Parameters

- base – LPUART peripheral base address
- txBuff – source buffer containing 8-bit data chars to send

- txSize – the number of bytes to send

Returns
An error code or lin_status_t

lin_status_t LIN_LPUART_GetTransmitStatus(LPUART_Type *base, uint8_t *bytesRemaining)

Get status of an on-going non-blocking transmission While sending frame data using non-blocking method, users can use this function to get status of that transmission. This function return LIN_TX_BUSY while sending, or LIN_TIMEOUT if timeout has occurred, or return LIN_SUCCESS when the transmission is complete. The bytesRemaining shows number of bytes that still needed to transmit.

Parameters

- base – LPUART peripheral base address
- bytesRemaining – Number of bytes still needed to transmit

Returns
lin_status_t LIN_TX_BUSY, LIN_SUCCESS or LIN_TIMEOUT

lin_status_t LIN_LPUART_RecvFrmDataBlocking(LPUART_Type *base, uint8_t *rxBuff, uint8_t rxSize, uint32_t timeoutMSec)

Receives frame data through the LIN_LPUART module using blocking method. This function will check the checksum byte. If the checksum is correct, it will receive the frame data. Blocking means that the function does not return until the reception is complete.

Parameters

- base – LPUART peripheral base address
- rxBuff – buffer containing 8-bit received data
- rxSize – the number of bytes to receive
- timeoutMSec – timeout value in milli seconds

Returns
An error code or lin_status_t

lin_status_t LIN_LPUART_RecvFrmData(LPUART_Type *base, uint8_t *rxBuff, uint8_t rxSize)

Receives frame data through the LIN_LPUART module using non-blocking method. This function will check the checksum byte. If the checksum is correct, it will receive it with the frame data. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the reception is complete.

Parameters

- base – LPUART peripheral base address
- rxBuff – buffer containing 8-bit received data
- rxSize – the number of bytes to receive

Returns
An error code or lin_status_t

lin_status_t LIN_LPUART_AbortTransferData(LPUART_Type *base)

Aborts an on-going non-blocking transmission/reception. While performing a non-blocking transferring data, users can call this function to terminate immediately the transferring.

Parameters

- base – LPUART peripheral base address

Returns
An error code or lin_status_t

lin_status_t LIN_LPUART_GetReceiveStatus(LPUART_Type *base, uint8_t *bytesRemaining)

Get status of an on-going non-blocking reception While receiving frame data using non-blocking method, users can use this function to get status of that receiving. This function return the current event ID, LIN_RX_BUSY while receiving and return LIN_SUCCESS, or time-out (LIN_TIMEOUT) when the reception is complete. The bytesRemaining shows number of bytes that still needed to receive.

Parameters

- base – LPUART peripheral base address
- bytesRemaining – Number of bytes still needed to receive

Returns

lin_status_t LIN_RX_BUSY, LIN_TIMEOUT or LIN_SUCCESS

lin_status_t LIN_LPUART_GoToSleepMode(LPUART_Type *base)

This function puts current node to sleep mode This function changes current node state to LIN_NODE_STATE_SLEEP_MODE.

Parameters

- base – LPUART peripheral base address

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_GotoIdleState(LPUART_Type *base)

Puts current LIN node to Idle state This function changes current node state to LIN_NODE_STATE_IDLE.

Parameters

- base – LPUART peripheral base address

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_SendWakeupSignal(LPUART_Type *base)

Sends a wakeup signal through the LIN_LPUART interface.

Parameters

- base – LPUART peripheral base address

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_MasterSendHeader(LPUART_Type *base, uint8_t id)

Sends frame header out through the LIN_LPUART module using a non-blocking method. This function sends LIN Break field, sync field then the ID with correct parity.

Parameters

- base – LPUART peripheral base address
- id – Frame Identifier

Returns

An error code or lin_status_t

lin_status_t LIN_LPUART_EnableIRQ(LPUART_Type *base)

Enables LIN_LPUART hardware interrupts.

Parameters

- base – LPUART peripheral base address

Returns
An error code or lin_status_t

lin_status_t LIN_LPUART_DisableIRQ(LPUART_Type *base)

Disables LIN_LPUART hardware interrupts.

Parameters

- base – LPUART peripheral base address

Returns
An error code or lin_status_t

lin_status_t LIN_LPUART_AutoBaudCapture(uint32_t instance)

This function capture bits time to detect break char, calculate baudrate from sync bits and enable transceiver if autobaud successful. This function should only be used in Slave. The timer should be in mode input capture of both rising and falling edges. The timer input capture pin should be externally connected to RXD pin.

Parameters

- instance – LPUART instance

Returns
lin_status_t

void LIN_LPUART_IRQHandler(LPUART_Type *base)

LIN_LPUART RX TX interrupt handler.

Parameters

- base – LPUART peripheral base address

LIN_LPUART_TRANSMISSION_COMPLETE_TIMEOUT

Max loops to wait for LPUART transmission complete.

When de-initializing the LIN LPUART module, the program shall wait for the previous transmission to complete. This parameter defines how many loops to check completion before return error. If defined as 0, driver will wait forever until completion.

AUTOBAUD_BAUDRATE_TOLERANCE

BIT_RATE_TOLERANCE_UNSYNC

BIT_DURATION_MAX_19200

BIT_DURATION_MIN_19200

BIT_DURATION_MAX_14400

BIT_DURATION_MIN_14400

BIT_DURATION_MAX_9600

BIT_DURATION_MIN_9600

BIT_DURATION_MAX_4800

BIT_DURATION_MIN_4800

BIT_DURATION_MAX_2400

BIT_DURATION_MIN_2400

TWO_BIT_DURATION_MAX_19200

TWO_BIT_DURATION_MIN_19200

TWO_BIT_DURATION_MAX_14400

TWO_BIT_DURATION_MIN_14400

TWO_BIT_DURATION_MAX_9600

TWO_BIT_DURATION_MIN_9600

TWO_BIT_DURATION_MAX_4800

TWO_BIT_DURATION_MIN_4800

TWO_BIT_DURATION_MAX_2400

TWO_BIT_DURATION_MIN_2400

AUTOBAUD_BREAK_TIME_MIN

2.30 LLWU: Low-Leakage Wakeup Unit Driver

static inline void LLWU_GetVersionId(LLWU_Type *base, *llwu_version_id_t* *versionId)

Gets the LLWU version ID.

This function gets the LLWU version ID, including the major version number, the minor version number, and the feature specification number.

Parameters

- base – LLWU peripheral base address.
- versionId – A pointer to the version ID structure.

static inline void LLWU_GetParam(LLWU_Type *base, *llwu_param_t* *param)

Gets the LLWU parameter.

This function gets the LLWU parameter, including a wakeup pin number, a module number, a DMA number, and a pin filter number.

Parameters

- base – LLWU peripheral base address.

- param – A pointer to the LLWU parameter structure.

void LLWU_SetExternalWakeupPinMode(LLWU_Type *base, uint32_t pinIndex, *llwu_external_pin_mode_t* pinMode)

Sets the external input pin source mode.

This function sets the external input pin source mode that is used as a wake up source.

Parameters

- base – LLWU peripheral base address.
- pinIndex – A pin index to be enabled as an external wakeup source starting from 1.
- pinMode – A pin configuration mode defined in the llwu_external_pin_modes_t.

bool LLWU_GetExternalWakeupPinFlag(LLWU_Type *base, uint32_t pinIndex)

Gets the external wakeup source flag.

This function checks the external pin flag to detect whether the MCU is woken up by the specific pin.

Parameters

- base – LLWU peripheral base address.
- pinIndex – A pin index, which starts from 1.

Returns

True if the specific pin is a wakeup source.

void LLWU_ClearExternalWakeupPinFlag(LLWU_Type *base, uint32_t pinIndex)

Clears the external wakeup source flag.

This function clears the external wakeup source flag for a specific pin.

Parameters

- base – LLWU peripheral base address.
- pinIndex – A pin index, which starts from 1.

static inline void LLWU_EnableInternalModuleInterruptWakup(LLWU_Type *base, uint32_t moduleIndex, bool enable)

Enables/disables the internal module source.

This function enables/disables the internal module source mode that is used as a wake up source.

Parameters

- base – LLWU peripheral base address.
- moduleIndex – A module index to be enabled as an internal wakeup source starting from 1.
- enable – An enable or a disable setting

static inline void LLWU_EnableInternalModuleDmaRequestWakup(LLWU_Type *base, uint32_t moduleIndex, bool enable)

Enables/disables the internal module DMA wakeup source.

This function enables/disables the internal DMA that is used as a wake up source.

Parameters

- base – LLWU peripheral base address.

- moduleIndex – An internal module index which is used as a DMA request source, starting from 1.
- enable – Enable or disable the DMA request source

void LLWU_SetPinFilterMode(LLWU_Type *base, uint32_t filterIndex, *llwu_external_pin_filter_mode_t* filterMode)

Sets the pin filter configuration.

This function sets the pin filter configuration.

Parameters

- base – LLWU peripheral base address.
- filterIndex – A pin filter index used to enable/disable the digital filter, starting from 1.
- filterMode – A filter mode configuration

bool LLWU_GetPinFilterFlag(LLWU_Type *base, uint32_t filterIndex)

Gets the pin filter configuration.

This function gets the pin filter flag.

Parameters

- base – LLWU peripheral base address.
- filterIndex – A pin filter index, which starts from 1.

Returns

True if the flag is a source of the existing low-leakage power mode.

void LLWU_ClearPinFilterFlag(LLWU_Type *base, uint32_t filterIndex)

Clears the pin filter configuration.

This function clears the pin filter flag.

Parameters

- base – LLWU peripheral base address.
- filterIndex – A pin filter index to clear the flag, starting from 1.

void LLWU_SetResetPinMode(LLWU_Type *base, bool pinEnable, bool pinFilterEnable)

Sets the reset pin mode.

This function determines how the reset pin is used as a low leakage mode exit source.

Parameters

- base – LLWU peripheral base address.
- pinEnable – Enable reset the pin filter
- pinFilterEnable – Specify whether the pin filter is enabled in Low-Leakage power mode.

FSL_LLWU_DRIVER_VERSION

LLWU driver version.

enum _llwu_external_pin_mode

External input pin control modes.

Values:

enumerator kLLWU_ExternalPinDisable
Pin disabled as a wakeup input.

enumerator kLLWU_ExternalPinRisingEdge
Pin enabled with the rising edge detection.

enumerator kLLWU_ExternalPinFallingEdge
Pin enabled with the falling edge detection.

enumerator kLLWU_ExternalPinAnyEdge
Pin enabled with any change detection.

enum _llwu_pin_filter_mode
Digital filter control modes.
Values:

enumerator kLLWU_PinFilterDisable
Filter disabled.

enumerator kLLWU_PinFilterRisingEdge
Filter positive edge detection.

enumerator kLLWU_PinFilterFallingEdge
Filter negative edge detection.

enumerator kLLWU_PinFilterAnyEdge
Filter any edge detection.

typedef enum *_llwu_external_pin_mode* llwu_external_pin_mode_t
External input pin control modes.

typedef enum *_llwu_pin_filter_mode* llwu_pin_filter_mode_t
Digital filter control modes.

typedef struct *_llwu_version_id* llwu_version_id_t
IP version ID definition.

typedef struct *_llwu_param* llwu_param_t
IP parameter definition.

typedef struct *_llwu_external_pin_filter_mode* llwu_external_pin_filter_mode_t
An external input pin filter control structure.

LLWU_REG_VAL(x)

struct _llwu_version_id
#include <fsl_llwu.h> IP version ID definition.

Public Members

uint16_t feature
A feature specification number.

uint8_t minor
The minor version number.

uint8_t major
The major version number.

struct __llwu_param
#include <fsl_llwu.h> IP parameter definition.

Public Members

uint8_t filters
A number of the pin filter.

uint8_t dmas
A number of the wakeup DMA.

uint8_t modules
A number of the wakeup module.

uint8_t pins
A number of the wake up pin.

struct __llwu_external_pin_filter_mode
#include <fsl_llwu.h> An external input pin filter control structure.

Public Members

uint32_t pinIndex
A pin number

llwu_pin_filter_mode_t filterMode
Filter mode

2.31 LPTMR: Low-Power Timer

void LPTMR_Init(LPTMR_Type *base, const *lptmr_config_t* *config)
Ungates the LPTMR clock and configures the peripheral for a basic operation.

Note: This API should be called at the beginning of the application using the LPTMR driver.

Parameters

- base – LPTMR peripheral base address
- config – A pointer to the LPTMR configuration structure.

void LPTMR_Deinit(LPTMR_Type *base)

Gates the LPTMR clock.

Parameters

- base – LPTMR peripheral base address

void LPTMR_GetDefaultConfig(*lptmr_config_t* *config)

Fills in the LPTMR configuration structure with default settings.

The default values are as follows.

```
config->timerMode = kLPTMR_TimerModeTimeCounter;
config->pinSelect = kLPTMR_PinSelectInput_0;
config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
config->enableFreeRunning = false;
config->bypassPrescaler = true;
config->prescalerClockSource = kLPTMR_PrescalerClock_1;
config->value = kLPTMR_Prescale_Glitch_0;
```

Parameters

- config – A pointer to the LPTMR configuration structure.

static inline void LPTMR_EnableInterrupts(LPTMR_Type *base, uint32_t mask)

Enables the selected LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *lptmr_interrupt_enable_t*

static inline void LPTMR_DisableInterrupts(LPTMR_Type *base, uint32_t mask)

Disables the selected LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address
- mask – The interrupts to disable. This is a logical OR of members of the enumeration *lptmr_interrupt_enable_t*.

static inline uint32_t LPTMR_GetEnabledInterrupts(LPTMR_Type *base)

Gets the enabled LPTMR interrupts.

Parameters

- base – LPTMR peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *lptmr_interrupt_enable_t*

static inline uint32_t LPTMR_GetStatusFlags(LPTMR_Type *base)

Gets the LPTMR status flags.

Parameters

- base – LPTMR peripheral base address

Returns
The status flags. This is the logical OR of members of the enumeration *lptmr_status_flags_t*

static inline void LPTMR_ClearStatusFlags(LPTMR_Type *base, uint32_t mask)

Clears the LPTMR status flags.

Parameters

- base – LPTMR peripheral base address
- mask – The status flags to clear. This is a logical OR of members of the enumeration *lptmr_status_flags_t*.

static inline void LPTMR_SetTimerPeriod(LPTMR_Type *base, uint32_t ticks)

Sets the timer period in units of count.

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note:

a. The TCF flag is set with the CNR equals the count provided here and then increments.

b. Call the utility macros provided in the fsl_common.h to convert to ticks.

Parameters

- base – LPTMR peripheral base address
- ticks – A timer period in units of ticks

static inline uint32_t LPTMR_GetCurrentTimerCount(LPTMR_Type *base)

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: Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

- base – LPTMR peripheral base address

Returns
The current counter value in ticks

static inline void LPTMR_StartTimer(LPTMR_Type *base)

Starts the timer.

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

- base – LPTMR peripheral base address

static inline void LPTMR_StopTimer(LPTMR_Type *base)

Stops the timer.

This function stops the timer and resets the timer's counter register.

Parameters

- base – LPTMR peripheral base address

FSL_LPTMR_DRIVER_VERSION
Driver Version

enum _lptmr_pin_select
LPTMR pin selection used in pulse counter mode.
Values:

enumerator kLPTMR_PinSelectInput_0
Pulse counter input 0 is selected

enumerator kLPTMR_PinSelectInput_1
Pulse counter input 1 is selected

enumerator kLPTMR_PinSelectInput_2
Pulse counter input 2 is selected

enumerator kLPTMR_PinSelectInput_3
Pulse counter input 3 is selected

enum _lptmr_pin_polarity
LPTMR pin polarity used in pulse counter mode.
Values:

enumerator kLPTMR_PinPolarityActiveHigh
Pulse Counter input source is active-high

enumerator kLPTMR_PinPolarityActiveLow
Pulse Counter input source is active-low

enum _lptmr_timer_mode
LPTMR timer mode selection.
Values:

enumerator kLPTMR_TimerModeTimeCounter
Time Counter mode

enumerator kLPTMR_TimerModePulseCounter
Pulse Counter mode

enum _lptmr_prescaler_glitch_value
LPTMR prescaler/glitch filter values.
Values:

enumerator kLPTMR_Prescale_Glitch_0
Prescaler divide 2, glitch filter does not support this setting

enumerator kLPTMR_Prescale_Glitch_1
Prescaler divide 4, glitch filter 2

enumerator kLPTMR_Prescale_Glitch_2
Prescaler divide 8, glitch filter 4

enumerator kLPTMR_Prescale_Glitch_3
Prescaler divide 16, glitch filter 8

enumerator kLPTMR_Prescale_Glitch_4
Prescaler divide 32, glitch filter 16

enumerator kLPTMR_Prescale_Glitch_5
Prescaler divide 64, glitch filter 32

enumerator kLPTMR_Prescale_Glitch_6
Prescaler divide 128, glitch filter 64

enumerator kLPTMR_Prescale_Glitch_7
Prescaler divide 256, glitch filter 128

enumerator kLPTMR_Prescale_Glitch_8
Prescaler divide 512, glitch filter 256

enumerator kLPTMR_Prescale_Glitch_9
Prescaler divide 1024, glitch filter 512

enumerator kLPTMR_Prescale_Glitch_10
Prescaler divide 2048 glitch filter 1024

enumerator kLPTMR_Prescale_Glitch_11
Prescaler divide 4096, glitch filter 2048

enumerator kLPTMR_Prescale_Glitch_12
Prescaler divide 8192, glitch filter 4096

enumerator kLPTMR_Prescale_Glitch_13
Prescaler divide 16384, glitch filter 8192

enumerator kLPTMR_Prescale_Glitch_14
Prescaler divide 32768, glitch filter 16384

enumerator kLPTMR_Prescale_Glitch_15
Prescaler divide 65536, glitch filter 32768

enum _lptmr_prescaler_clock_select
LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

Values:

enumerator kLPTMR_PrescalerClock_0
Prescaler/glitch filter clock 0 selected.

enumerator kLPTMR_PrescalerClock_1
Prescaler/glitch filter clock 1 selected.

enumerator kLPTMR_PrescalerClock_2
Prescaler/glitch filter clock 2 selected.

enumerator kLPTMR_PrescalerClock_3
Prescaler/glitch filter clock 3 selected.

enum _lptmr_interrupt_enable
List of the LPTMR interrupts.
Values:

enumerator kLPTMR_TimerInterruptEnable
Timer interrupt enable

enum _lptmr_status_flags
List of the LPTMR status flags.
Values:

enumerator kLPTMR_TimerCompareFlag
Timer compare flag

typedef enum *_lptmr_pin_select* lptmr_pin_select_t
LPTMR pin selection used in pulse counter mode.

typedef enum *_lptmr_pin_polarity* lptmr_pin_polarity_t
LPTMR pin polarity used in pulse counter mode.

typedef enum *_lptmr_timer_mode* lptmr_timer_mode_t
LPTMR timer mode selection.

typedef enum *_lptmr_prescaler_glitch_value* lptmr_prescaler_glitch_value_t
LPTMR prescaler/glitch filter values.

typedef enum *_lptmr_prescaler_clock_select* lptmr_prescaler_clock_select_t
LPTMR prescaler/glitch filter clock select.

Note: Clock connections are SoC-specific

typedef enum *_lptmr_interrupt_enable* lptmr_interrupt_enable_t

List of the LPTMR interrupts.

typedef enum *_lptmr_status_flags* lptmr_status_flags_t

List of the LPTMR status flags.

typedef struct *_lptmr_config* lptmr_config_t

LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the *LPTMR_GetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

static inline void LPTMR_EnableTimerDMA(LPTMR_Type *base, bool enable)

Enable or disable timer DMA request.

Toggles CSR[TDRE] on the LPTMR side only: when enabled, every compare event (CSR[TCF]=1) raises an LPTMR DMA request line which is auto-cleared after the DMA controller services it.

Note: LPTMR runs on a low-power clock that is asynchronous to the EDMA bus clock. On some EDMA IP variants the channel has TWO gates that must BOTH be opened for an LPTMR request to be accepted: the base hardware request gate (ERQ, opened by *EDMA_EnableChannelRequest()*) and an extra asynchronous-request gate (opened by *EDMA_EnableAsyncRequest()* on EDMA4; the classic EDMA + DMAMUX combination does not need this).

Parameters

- base – base LPTMR peripheral base address
- enable – Switcher of timer DMA feature. “true” means to enable, “false” means to disable.

struct _lptmr_config

#include <fsl_lptmr.h> LPTMR config structure.

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the *LPTMR_GetDefaultConfig()* function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

Public Members

lptmr_timer_mode_t timerMode

Time counter mode or pulse counter mode

lptmr_pin_select_t pinSelect

LPTMR pulse input pin select; used only in pulse counter mode

lptmr_pin_polarity_t pinPolarity

LPTMR pulse input pin polarity; used only in pulse counter mode

bool enableFreeRunning

True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set

bool bypassPrescaler

True: bypass prescaler; false: use clock from prescaler

lptmr_prescaler_clock_select_t prescalerClockSource

LPTMR clock source

lptmr_prescaler_glitch_value_t value

Prescaler or glitch filter value

2.32 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

2.33 LPUART Driver

static inline void LPUART_SoftwareReset(LPUART_Type *base)

Resets the LPUART using software.

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

Parameters

- base – LPUART peripheral base address.

status_t LPUART_Init(LPUART_Type *base, const *lpuart_config_t* *config, uint32_t srcClock_Hz)

Initializes an LPUART instance with the user configuration structure and the peripheral clock.

This function configures the LPUART module with user-defined settings. Call the *LPUART_GetDefaultConfig()* function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
lpuart_config_t lpuartConfig;
lpuartConfig.baudRate_Bps = 115200U;
lpuartConfig.parityMode = kLPUART_ParityDisabled;
lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
lpuartConfig.isMsb = false;
lpuartConfig.stopBitCount = kLPUART_OneStopBit;
lpuartConfig.txFifoWatermark = 0;
lpuartConfig.rxFifoWatermark = 1;
LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- config – Pointer to a user-defined configuration structure.

- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – LPUART initialize succeed

status_t LPUART_Deinit(LPUART_Type *base)

Deinitializes a LPUART instance.

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

- base – LPUART peripheral base address.

Return values

- kStatus_Success – Deinit is success.
- kStatus_LPUART_Timeout – Timeout during deinit.

void LPUART_GetDefaultConfig(*lpuart_config_t* *config)

Gets the default configuration structure.

This function initializes the LPUART configuration structure to a default value. The default values are: lpuartConfig->baudRate_Bps = 115200U; lpuartConfig->parityMode = kLPUART_ParityDisabled; lpuartConfig->dataBitsCount = kLPUART_EightDataBits; lpuartConfig->isMsb = false; lpuartConfig->stopBitCount = kLPUART_OneStopBit; lpuartConfig->txFifoWatermark = 0; lpuartConfig->rxFifoWatermark = 1; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

- config – Pointer to a configuration structure.

status_t LPUART_SetBaudRate(LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the LPUART instance baudrate.

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
```

Parameters

- base – LPUART peripheral base address.
- baudRate_Bps – LPUART baudrate to be set.
- srcClock_Hz – LPUART clock source frequency in HZ.

Return values

- kStatus_LPUART_BaudrateNotSupport – Baudrate is not supported in the current clock source.
- kStatus_Success – Set baudrate succeeded.

void LPUART_Enable9bitMode(LPUART_Type *base, bool enable)

Enable 9-bit data mode for LPUART.

This function set the 9-bit mode for LPUART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- base – LPUART peripheral base address.
- enable – true to enable, flase to disable.

static inline void LPUART_SetMatchAddress(LPUART_Type *base, uint16_t address1, uint16_t address2)

Set the LPUART address.

This function configures the address for LPUART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any LPUART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

- base – LPUART peripheral base address.
- address1 – LPUART slave address1.
- address2 – LPUART slave address2.

static inline void LPUART_EnableMatchAddress(LPUART_Type *base, bool match1, bool match2)

Enable the LPUART match address feature.

Parameters

- base – LPUART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void LPUART_SetRxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Rx FIFO watermark.

static inline void LPUART_SetTxFifoWatermark(LPUART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – LPUART peripheral base address.
- water – Tx FIFO watermark.

static inline void LPUART_TransferEnable16Bit(*lpuart_handle_t* *handle, bool enable)

Sets the LPUART using 16bit transmit, only for 9bit or 10bit mode.

This function Enable 16bit Data transmit in lpuart_handle_t.

Parameters

- handle – LPUART handle pointer.
- enable – true to enable, false to disable.

uint32_t LPUART_GetStatusFlags(LPUART_Type *base)

Gets LPUART status flags.

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators *_lpuart_flags*. To check for a specific status, compare the return value with enumerators in the *_lpuart_flags*. For example, to check whether the TX is empty:

```
if (kLPUART_TxDataRegEmptyFlag & LPUART_GetStatusFlags(LPUART1))
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the _lpuart_flags.

status_t LPUART_ClearStatusFlags(LPUART_Type *base, uint32_t mask)

Clears status flags with a provided mask.

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only cleared or set by hardware are: kLPUART_TxDataRegEmptyFlag, kLPUART_TransmissionCompleteFlag, kLPUART_RxDataRegFullFlag, kLPUART_RxActiveFlag, kLPUART_NoiseErrorFlag, kLPUART_ParityErrorFlag, kLPUART_TxFifoEmptyFlag,kLPUART_RxFifoEmptyFlag Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

Parameters

- base – LPUART peripheral base address.
- mask – the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask.

Return values

- kStatus_LPUART_FlagCannotClearManually – The flag can't be cleared by this function but it is cleared automatically by hardware.
- kStatus_Success – Status in the mask are cleared.

Returns

0 succeed, others failed.

void LPUART_EnableInterrupts(LPUART_Type *base, uint32_t mask)

Enables LPUART interrupts according to a provided mask.

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the *_lpuart_interrupt_enable*. This examples shows how to enable TX empty interrupt and RX full interrupt:

```
LPUART_EnableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to enable. Logical OR of *_lpuart_interrupt_enable*.

void LPUART_DisableInterrupts(LPUART_Type *base, uint32_t mask)

Disables LPUART interrupts according to a provided mask.

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See *_lpuart_interrupt_enable*. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
LPUART_DisableInterrupts(LPUART1,kLPUART_TxDataRegEmptyInterruptEnable | kLPUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – LPUART peripheral base address.
- mask – The interrupts to disable. Logical OR of *_lpuart_interrupt_enable*.

uint32_t LPUART_GetEnabledInterrupts(LPUART_Type *base)

Gets enabled LPUART interrupts.

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators *_lpuart_interrupt_enable*. To check a specific interrupt enable status, compare the return value with enumerators in *_lpuart_interrupt_enable*. For example, to check whether the TX empty interrupt is enabled:

```
uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);

if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in *_lpuart_interrupt_enable*.

static inline uintptr_t LPUART_GetDataRegisterAddress(LPUART_Type *base)

Gets the LPUART data register address.

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

static inline void LPUART_EnableTxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter DMA request.

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRxDMA(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver DMA.

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

uint32_t LPUART_GetInstance(LPUART_Type *base)

Get the LPUART instance from peripheral base address.

Parameters

- base – LPUART peripheral base address.

Returns

LPUART instance.

static inline void LPUART_EnableTx(LPUART_Type *base, bool enable)

Enables or disables the LPUART transmitter.

This function enables or disables the LPUART transmitter.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_EnableRx(LPUART_Type *base, bool enable)

Enables or disables the LPUART receiver.

This function enables or disables the LPUART receiver.

Parameters

- base – LPUART peripheral base address.
- enable – True to enable, false to disable.

static inline void LPUART_WriteByte(LPUART_Type *base, uint8_t data)

Writes to the transmitter register.

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

- base – LPUART peripheral base address.
- data – Data write to the TX register.

static inline uint8_t LPUART_ReadByte(LPUART_Type *base)

Reads the receiver register.

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

- base – LPUART peripheral base address.

Returns
Data read from data register.

static inline uint8_t LPUART_GetRxFifoCount(LPUART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns
rx FIFO data count.

static inline uint8_t LPUART_GetTxFifoCount(LPUART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – LPUART peripheral base address.

Returns
tx FIFO data count.

void LPUART_SendAddress(LPUART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

Parameters

- base – LPUART peripheral base address.
- address – LPUART slave address.

status_t LPUART_WriteBlocking(LPUART_Type *base, const uint8_t *data, size_t length)

Writes to the transmitter register using a blocking method.

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the dat to be sent out to the bus.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_WriteBlocking16bit(LPUART_Type *base, const uint16_t *data, size_t length)

Writes to the transmitter register using a blocking method in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer. Please make sure only 10bit of data is valid and other bits are 0.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t LPUART_ReadBlocking(LPUART_Type *base, uint8_t *data, size_t length)

Reads the receiver data register using a blocking method.

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

status_t LPUART_ReadBlocking16bit(LPUART_Type *base, uint16_t *data, size_t length)

Reads the receiver data register in 9bit or 10bit mode.

Note: This function only support 9bit or 10bit transfer.

Parameters

- base – LPUART peripheral base address.
- data – Start address of the buffer to store the received data by 16bit, only 10bit is valid.
- length – Size of the buffer.

Return values

- kStatus_LPUART_RxHardwareOverrun – Receiver overrun happened while receiving data.
- kStatus_LPUART_NoiseError – Noise error happened while receiving data.
- kStatus_LPUART_FramingError – Framing error happened while receiving data.
- kStatus_LPUART_ParityError – Parity error happened while receiving data.
- kStatus_LPUART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

void LPUART_TransferCreateHandle(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_callback_t* callback, void *userData)

Initializes the LPUART handle.

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the *LPUART_TransferReceiveNonBlocking()* API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as ringBuffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- callback – Callback function.
- userData – User data.

status_t LPUART_TransferSendNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the *kStatus_LPUART_TxIdle* as status parameter.

Note: The kStatus_LPUART_TxIdle is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the TX, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see *lpuart_transfer_t*.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_LPUART_TxBusy – Previous transmission still not finished, data not all written to the TX register.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferStartRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the *UART_TransferReceiveNonBlocking()* API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note: When using RX ring buffer, one byte is reserved for internal use. In other words, if ringBufferSize is 32, then only 31 bytes are used for saving data.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- ringBuffer – Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
- ringBufferSize – size of the ring buffer.

void LPUART_TransferStopRingBuffer(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

size_t LPUART_TransferGetRxRingBufferLength(LPUART_Type *base, *lpuart_handle_t* *handle)

Get the length of received data in RX ring buffer.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

void LPUART_TransferAbortSend(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetSendCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been sent out to bus.

This function gets the number of bytes that have been sent out to bus by an interrupt method.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.

- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferReceiveNonBlocking(LPUART_Type *base, *lpuart_handle_t* *handle, *lpuart_transfer_t* *xfer, size_t *receivedBytes)

Receives a buffer of data using the interrupt method.

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter receivedBytes shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter kStatus_UART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to xfer->data, which returns with the parameter receivedBytes set to 5. For the remaining 5 bytes, the newly arrived data is saved from xfer->data[5]. When 5 bytes are received, the LPUART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to xfer->data. When all data is received, the upper layer is notified.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART transfer structure, see uart_transfer_t.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into the transmit queue.
- kStatus_LPUART_RxBusy – Previous receive request is not finished.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortReceive(LPUART_Type *base, *lpuart_handle_t* *handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.

status_t LPUART_TransferGetReceiveCount(LPUART_Type *base, *lpuart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.

- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferHandleIRQ(LPUART_Type *base, void *irqHandle)

LPUART IRQ handle function.

This function handles the LPUART transmit and receive IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_TransferHandleErrorIRQ(LPUART_Type *base, void *irqHandle)

LPUART Error IRQ handle function.

This function handles the LPUART error IRQ request.

Parameters

- base – LPUART peripheral base address.
- irqHandle – LPUART handle pointer.

void LPUART_DriverIRQHandler(uint32_t instance)

LPUART driver IRQ handler common entry.

This function provides the common IRQ request entry for LPUART.

Parameters

- instance – LPUART instance.

FSL_LPUART_DRIVER_VERSION

LPUART driver version.

Error codes for the LPUART driver.

Values:

enumerator kStatus_LPUART_TxBusy

TX busy

enumerator kStatus_LPUART_RxBusy

RX busy

enumerator kStatus_LPUART_TxIdle

LPUART transmitter is idle.

enumerator kStatus_LPUART_RxIdle

LPUART receiver is idle.

enumerator kStatus_LPUART_TxWatermarkTooLarge

TX FIFO watermark too large

enumerator kStatus_LPUART_RxWatermarkTooLarge

RX FIFO watermark too large

enumerator kStatus_LPUART_FlagCannotClearManually
Some flag can't manually clear

enumerator kStatus_LPUART_Error
Error happens on LPUART.

enumerator kStatus_LPUART_RxRingBufferOverrun
LPUART RX software ring buffer overrun.

enumerator kStatus_LPUART_RxHardwareOverrun
LPUART RX receiver overrun.

enumerator kStatus_LPUART_NoiseError
LPUART noise error.

enumerator kStatus_LPUART_FramingError
LPUART framing error.

enumerator kStatus_LPUART_ParityError
LPUART parity error.

enumerator kStatus_LPUART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_LPUART_IdleLineDetected
IDLE flag.

enumerator kStatus_LPUART_Timeout
LPUART times out.

enum _lpuart_parity_mode
LPUART parity mode.
Values:

enumerator kLPUART_ParityDisabled
Parity disabled

enumerator kLPUART_ParityEven
Parity enabled, type even, bit setting: PE|PT = 10

enumerator kLPUART_ParityOdd
Parity enabled, type odd, bit setting: PE|PT = 11

enum _lpuart_data_bits
LPUART data bits count.
Values:

enumerator kLPUART_EightDataBits
Eight data bit

enumerator kLPUART_SevenDataBits
Seven data bit

enum _lpuart_stop_bit_count
LPUART stop bit count.
Values:

enumerator kLPUART_OneStopBit
One stop bit

enumerator kLPUART_TwoStopBit
Two stop bits

enum _lpuart_transmit_cts_source
LPUART transmit CTS source.
Values:

enumerator kLPUART_CtsSourcePin
CTS resource is the LPUART_CTS pin.

enumerator kLPUART_CtsSourceMatchResult
CTS resource is the match result.

enum _lpuart_transmit_cts_config
LPUART transmit CTS configure.
Values:

enumerator kLPUART_CtsSampleAtStart
CTS input is sampled at the start of each character.

enumerator kLPUART_CtsSampleAtIdle
CTS input is sampled when the transmitter is idle

enum _lpuart_transmit_rts_polarity
LPUART transmitter RTS polarity.
Values:

enumerator kLPUART_RtsPolarityLow
Transmitter RTS is active low.

enumerator kLPUART_RtsPolarityHigh
Transmitter RTS is active high.

enum _lpuart_idle_type_select

LPUART idle flag type defines when the receiver starts counting.

Values:

enumerator kLPUART_IdleTypeStartBit

Start counting after a valid start bit.

enumerator kLPUART_IdleTypeStopBit

Start counting after a stop bit.

enum _lpuart_idle_config

LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

Values:

enumerator kLPUART_IdleCharacter1

the number of idle characters.

enumerator kLPUART_IdleCharacter2

the number of idle characters.

enumerator kLPUART_IdleCharacter4

the number of idle characters.

enumerator kLPUART_IdleCharacter8

the number of idle characters.

enumerator kLPUART_IdleCharacter16

the number of idle characters.

enumerator kLPUART_IdleCharacter32

the number of idle characters.

enumerator kLPUART_IdleCharacter64

the number of idle characters.

enumerator kLPUART_IdleCharacter128

the number of idle characters.

enum _lpuart_interrupt_enable

LPUART interrupt configuration structure, default settings all disabled.

This structure contains the settings for all LPUART interrupt configurations.

Values:

enumerator kLPUART_LinBreakInterruptEnable

LIN break detect. bit 7

enumerator kLPUART_RxActiveEdgeInterruptEnable
Receive Active Edge. bit 6

enumerator kLPUART_TxDataRegEmptyInterruptEnable
Transmit data register empty. bit 23

enumerator kLPUART_TransmissionCompleteInterruptEnable
Transmission complete. bit 22

enumerator kLPUART_RxDataRegFullInterruptEnable
Receiver data register full. bit 21

enumerator kLPUART_IdleLineInterruptEnable
Idle line. bit 20

enumerator kLPUART_RxOverrunInterruptEnable
Receiver Overrun. bit 27

enumerator kLPUART_NoiseErrorInterruptEnable
Noise error flag. bit 26

enumerator kLPUART_FramingErrorInterruptEnable
Framing error flag. bit 25

enumerator kLPUART_ParityErrorInterruptEnable
Parity error flag. bit 24

enumerator kLPUART_Match1InterruptEnable
Parity error flag. bit 15

enumerator kLPUART_Match2InterruptEnable
Parity error flag. bit 14

enumerator kLPUART_TxFifoOverflowInterruptEnable
Transmit FIFO Overflow. bit 9

enumerator kLPUART_RxFifoUnderflowInterruptEnable
Receive FIFO Underflow. bit 8

enumerator kLPUART_AllInterruptEnable

enum _lpuart_flags
LPUART status flags.
This provides constants for the LPUART status flags for use in the LPUART functions.
Values:

enumerator kLPUART_TxDataRegEmptyFlag
Transmit data register empty flag, sets when transmit buffer is empty. bit 23

enumerator kLPUART_TransmissionCompleteFlag

Transmission complete flag, sets when transmission activity complete. bit 22

enumerator kLPUART_RxDataRegFullFlag

Receive data register full flag, sets when the receive data buffer is full. bit 21

enumerator kLPUART_IdleLineFlag

Idle line detect flag, sets when idle line detected. bit 20

enumerator kLPUART_RxOverrunFlag

Receive Overrun, sets when new data is received before data is read from receive register. bit 19

enumerator kLPUART_NoiseErrorFlag

Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets. bit 18

enumerator kLPUART_FramingErrorFlag

Frame error flag, sets if logic 0 was detected where stop bit expected. bit 17

enumerator kLPUART_ParityErrorFlag

If parity enabled, sets upon parity error detection. bit 16

enumerator kLPUART_LinBreakFlag

LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled. bit 31

enumerator kLPUART_RxActiveEdgeFlag

Receive pin active edge interrupt flag, sets when active edge detected. bit 30

enumerator kLPUART_RxActiveFlag

Receiver Active Flag (RAF), sets at beginning of valid start. bit 24

enumerator kLPUART_DataMatch1Flag

The next character to be read from LPUART_DATA matches MA1. bit 15

enumerator kLPUART_DataMatch2Flag

The next character to be read from LPUART_DATA matches MA2. bit 14

enumerator kLPUART_TxFifoEmptyFlag

TXEMPT bit, sets if transmit buffer is empty. bit 7

enumerator kLPUART_RxFifoEmptyFlag

RXEMPT bit, sets if receive buffer is empty. bit 6

enumerator kLPUART_TxFifoOverflowFlag

TXOF bit, sets if transmit buffer overflow occurred. bit 1

enumerator kLPUART_RxFifoUnderflowFlag
RXUF bit, sets if receive buffer underflow occurred. bit 0

enumerator kLPUART_AllClearFlags

enumerator kLPUART_AllFlags

typedef enum *_lpuart_parity_mode* lpuart_parity_mode_t
LPUART parity mode.

typedef enum *_lpuart_data_bits* lpuart_data_bits_t
LPUART data bits count.

typedef enum *_lpuart_stop_bit_count* lpuart_stop_bit_count_t
LPUART stop bit count.

typedef enum *_lpuart_transmit_cts_source* lpuart_transmit_cts_source_t
LPUART transmit CTS source.

typedef enum *_lpuart_transmit_cts_config* lpuart_transmit_cts_config_t
LPUART transmit CTS configure.

typedef enum *_lpuart_transmit_rts_polarity* lpuart_transmit_rts_polarity_t
LPUART transmitter RTS polarity.

typedef enum *_lpuart_idle_type_select* lpuart_idle_type_select_t
LPUART idle flag type defines when the receiver starts counting.

typedef enum *_lpuart_idle_config* lpuart_idle_config_t
LPUART idle detected configuration. This structure defines the number of idle characters that must be received before the IDLE flag is set.

typedef struct *_lpuart_config* lpuart_config_t
LPUART configuration structure.

typedef struct *_lpuart_transfer* lpuart_transfer_t
LPUART transfer structure.

typedef struct *_lpuart_handle* lpuart_handle_t

typedef void (*lpuart_transfer_callback_t)(LPUART_Type *base, *lpuart_handle_t* *handle, *status_t* status, void *userData)
LPUART transfer callback function.

typedef void (*lpuart_isr_t)(LPUART_Type *base, void *handle)

void *s_lpuartHandle[]

const IRQn_Type s_lpuartTxIRQ[]

lpuart_isr_t s_lpuartIsr[]

UART_RETRY_TIMES
Retry times for waiting flag.

struct _lpuart_config
#include <fsl_lpuart.h> LPUART configuration structure.

Public Members

uint32_t baudRate_Bps
LPUART baud rate

lpuart_parity_mode_t parityMode
Parity mode, disabled (default), even, odd

lpuart_data_bits_t dataBitsCount
Data bits count, eight (default), seven

bool isMsb
Data bits order, LSB (default), MSB

lpuart_stop_bit_count_t stopBitCount
Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark
TX FIFO watermark

uint8_t rxFifoWatermark
RX FIFO watermark

bool enableRxRTS
RX RTS enable

bool enableTxRTS
TX RTS enable

bool enableTxCTS
TX CTS enable

lpuart_transmit_cts_source_t txCtsSource
TX CTS source

lpuart_transmit_cts_config_t txCtsConfig
TX CTS configure

lpuart_transmit_rts_polarity_t txRtsPolarity
TX RTS polarity

lpuart_idle_type_select_t rxIdleType
RX IDLE type.

lpuart_idle_config_t rxIdleConfig
RX IDLE configuration.

bool enableTx
Enable TX

bool enableRx
Enable RX

bool inverseTxd
Transmit Data Inversion - Setting true reverses the polarity of the transmitted data output

struct __lpuart__transfer
#include <fsl_lpuart.h> LPUART transfer structure.

Public Members

size_t dataSize
The byte count to be transfer.

struct __lpuart__handle
#include <fsl_lpuart.h> LPUART handle structure.

Public Members

volatile size_t txDataSize
Size of the remaining data to send.

size_t txDataSizeAll
Size of the data to send out.

volatile size_t rxDataSize
Size of the remaining data to receive.

size_t rxDataSizeAll
Size of the data to receive.

size_t rxRingBufferSize
Size of the ring buffer.

volatile uint16_t rxRingBufferHead
Index for the driver to store received data into ring buffer.

volatile uint16_t rxRingBufferTail
Index for the user to get data from the ring buffer.

lpuart_transfer_callback_t callback
Callback function.

void *userData
LPUART callback function parameter.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state.

bool isSevenDataBits
Seven data bits flag.

bool is16bitData
16bit data bits flag, only used for 9bit or 10bit data

union ___unnamed21___

Public Members

uint8_t *data
The buffer of data to be transfer.

uint8_t *rxData
The buffer to receive data.

uint16_t *rxData16
The buffer to receive data.

const uint8_t *txData
The buffer of data to be sent.

const uint16_t *txData16
The buffer of data to be sent.

union ___unnamed23___

Public Members

const uint8_t *volatile txData
Address of remaining data to send.

const uint16_t *volatile txData16
Address of remaining data to send.

union ___unnamed25___

Public Members

uint8_t *volatile rxData
Address of remaining data to receive.

uint16_t *volatile rxData16
Address of remaining data to receive.

union ___unnamed27___

Public Members

uint8_t *rxRingBuffer
Start address of the receiver ring buffer.

uint16_t *rxRingBuffer16
Start address of the receiver ring buffer.

2.34 LPUART eDMA Driver

void LPUART_TransferCreateHandleEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle)

Initializes the LPUART handle which is used in transactional functions.

Note: This function disables all LPUART interrupts.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- callback – Callback function.
- userData – User data.
- txEdmaHandle – User requested DMA handle for TX DMA transfer.

- rxEdmaHandle – User requested DMA handle for RX DMA transfer.

status_t LPUART_SendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- xfer – LPUART eDMA transfer structure. See *lpuart_transfer_t*.

Return values

- kStatus_Success – if succeed, others failed.
- kStatus_LPUART_TxBusy – Previous transfer on going.
- kStatus_InvalidArgument – Invalid argument.

status_t LPUART_ReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *lpuart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.
- xfer – LPUART eDMA transfer structure, see *lpuart_transfer_t*.

Return values

- kStatus_Success – if succeed, others fail.
- kStatus_LPUART_RxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

void LPUART_TransferAbortSendEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the sent data using eDMA.

This function aborts the sent data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

void LPUART_TransferAbortReceiveEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle)

Aborts the received data using eDMA.

This function aborts the received data using eDMA.

Parameters

- base – LPUART peripheral base address.
- handle – Pointer to lpuart_edma_handle_t structure.

status_t LPUART_TransferGetSendCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of bytes written to the LPUART TX register.

This function gets the number of bytes written to the LPUART TX register by DMA.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t LPUART_TransferGetReceiveCountEDMA(LPUART_Type *base, *lpuart_edma_handle_t* *handle, uint32_t *count)

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- base – LPUART peripheral base address.
- handle – LPUART handle pointer.
- count – Receive bytes count.

Return values

- kStatus_NoTransferInProgress – No receive in progress.
- kStatus_InvalidArgument – Parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

void LPUART_TransferEdmaHandleIRQ(LPUART_Type *base, void *lpuartEdmaHandle)

LPUART eDMA IRQ handle function.

This function handles the LPUART tx complete IRQ request and invoke user callback. It is not set to static so that it can be used in user application.

Note: This function is used as default IRQ handler by double weak mechanism. If user's specific IRQ handler is implemented, make sure this function is invoked in the handler.

Parameters

- base – LPUART peripheral base address.
- lpuartEdmaHandle – LPUART handle pointer.

FSL_LPUART_EDMA_DRIVER_VERSION

LPUART EDMA driver version.

typedef struct *_lpuart_edma_handle* lpuart_edma_handle_t

typedef void (*lpuart_edma_transfer_callback_t)(LPUART_Type *base, *lpuart_edma_handle_t* *handle, *status_t* status, void *userData)

LPUART transfer callback function.

struct _lpuart_edma_handle

#include <fsl_lpuart_edma.h> LPUART eDMA handle.

Public Members

lpuart_edma_transfer_callback_t callback

Callback function.

void *userData

LPUART callback function parameter.

size_t rxDataSizeAll

Size of the data to receive.

size_t txDataSizeAll

Size of the data to send out.

edma_handle_t *txEdmaHandle

The eDMA TX channel used.

edma_handle_t *rxEdmaHandle

The eDMA RX channel used.

uint8_t nbytes

eDMA minor byte transfer count initially configured.

volatile uint8_t txState

TX transfer state.

volatile uint8_t rxState

RX transfer state

2.35 MCM: Miscellaneous Control Module

FSL_MCM_DRIVER_VERSION

MCM driver version.

Enum _mcm_interrupt_flag. Interrupt status flag mask. .

Values:

enumerator kMCM_CacheWriteBuffer
Cache Write Buffer Error Enable.

enumerator kMCM_ParityError
Cache Parity Error Enable.

enumerator kMCM_FPUInvalidOperation
FPU Invalid Operation Interrupt Enable.

enumerator kMCM_FPUDivideByZero
FPU Divide-by-zero Interrupt Enable.

enumerator kMCM_FPUOverflow
FPU Overflow Interrupt Enable.

enumerator kMCM_FPUUnderflow
FPU Underflow Interrupt Enable.

enumerator kMCM_FPUInexact
FPU Inexact Interrupt Enable.

enumerator kMCM_FPUInputDenormalInterrupt
FPU Input Denormal Interrupt Enable.

typedef union *_mcm_buffer_fault_attribute* mcm_buffer_fault_attribute_t
The union of buffer fault attribute.

typedef union *_mcm_lmem_fault_attribute* mcm_lmem_fault_attribute_t
The union of LMEM fault attribute.

static inline void MCM_EnableCrossbarRoundRobin(MCM_Type *base, bool enable)
Enables/Disables crossbar round robin.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable crossbar round robin.
  - **true** Enable crossbar round robin.
  - **false** disable crossbar round robin.

static inline void MCM_EnableInterruptStatus(MCM_Type *base, uint32_t mask)
Enables the interrupt.

Parameters

- base – MCM peripheral base address.
- mask – Interrupt status flags mask(*_mcm_interrupt_flag*).

static inline void MCM_DisableInterruptStatus(MCM_Type *base, uint32_t mask)
Disables the interrupt.

Parameters

- base – MCM peripheral base address.

- mask – Interrupt status flags mask(*_mcm_interrupt_flag*).

static inline uint16_t MCM_GetInterruptStatus(MCM_Type *base)

Gets the Interrupt status .

Parameters

- base – MCM peripheral base address.

static inline void MCM_ClearCacheWriteBufferErroStatus(MCM_Type *base)

Clears the Interrupt status .

Parameters

- base – MCM peripheral base address.

static inline uint32_t MCM_GetBufferFaultAddress(MCM_Type *base)

Gets buffer fault address.

Parameters

- base – MCM peripheral base address.

static inline void MCM_GetBufferFaultAttribute(MCM_Type *base, *mcm_buffer_fault_attribute_t* *bufferfault)

Gets buffer fault attributes.

Parameters

- base – MCM peripheral base address.
- bufferfault – Structure to store the result.

static inline uint32_t MCM_GetBufferFaultData(MCM_Type *base)

Gets buffer fault data.

Parameters

- base – MCM peripheral base address.

static inline void MCM_LimitCodeCachePeripheralWriteBuffering(MCM_Type *base, bool enable)

Limit code cache peripheral write buffering.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable limit code cache peripheral write buffering.
  - **true** Enable limit code cache peripheral write buffering.
  - **false** disable limit code cache peripheral write buffering.

static inline void MCM_BypassFixedCodeCacheMap(MCM_Type *base, bool enable)

Bypass fixed code cache map.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable bypass fixed code cache map.
  - **true** Enable bypass fixed code cache map.
  - **false** disable bypass fixed code cache map.

static inline void MCM_EnableCodeBusCache(MCM_Type *base, bool enable)

Enables/Disables code bus cache.

Parameters

- base – MCM peripheral base address.

- enable – Used to disable/enable code bus cache.
  - **true** Enable code bus cache.
  - **false** disable code bus cache.

static inline void MCM_ForceCodeCacheToNoAllocation(MCM_Type *base, bool enable)

Force code cache to no allocation.

Parameters

- base – MCM peripheral base address.
- enable – Used to force code cache to allocation or no allocation.
  - **true** Force code cache to no allocation.
  - **false** Force code cache to allocation.

static inline void MCM_EnableCodeCacheWriteBuffer(MCM_Type *base, bool enable)

Enables/Disables code cache write buffer.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable code cache write buffer.
  - **true** Enable code cache write buffer.
  - **false** Disable code cache write buffer.

static inline void MCM_ClearCodeBusCache(MCM_Type *base)

Clear code bus cache.

Parameters

- base – MCM peripheral base address.

static inline void MCM_EnablePcParityFaultReport(MCM_Type *base, bool enable)

Enables/Disables PC Parity Fault Report.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity Fault Report.
  - **true** Enable PC Parity Fault Report.
  - **false** disable PC Parity Fault Report.

static inline void MCM_EnablePcParity(MCM_Type *base, bool enable)

Enables/Disables PC Parity.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable PC Parity.
  - **true** Enable PC Parity.
  - **false** disable PC Parity.

static inline void MCM_LockConfigState(MCM_Type *base)

Lock the configuration state.

Parameters

- base – MCM peripheral base address.

static inline void MCM_EnableCacheParityReporting(MCM_Type *base, bool enable)

Enables/Disables cache parity reporting.

Parameters

- base – MCM peripheral base address.
- enable – Used to enable/disable cache parity reporting.
  - **true** Enable cache parity reporting.
  - **false** disable cache parity reporting.

static inline uint32_t MCM_GetLmemFaultAddress(MCM_Type *base)

Gets LMEM fault address.

Parameters

- base – MCM peripheral base address.

static inline void MCM_GetLmemFaultAttribute(MCM_Type *base, *mcm_lmem_fault_attribute_t* *lmemFault)

Get LMEM fault attributes.

Parameters

- base – MCM peripheral base address.
- lmemFault – Structure to store the result.

static inline uint64_t MCM_GetLmemFaultData(MCM_Type *base)

Gets LMEM fault data.

Parameters

- base – MCM peripheral base address.

MCM_LMFATR_TYPE_MASK

MCM_LMFATR_MODE_MASK

MCM_LMFATR_BUFF_MASK

MCM_LMFATR_CACH_MASK

MCM_ISCR_STAT_MASK

FSL_COMPONENT_ID

union __mcm_buffer_fault_attribute

#include <fsl_mcm.h> The union of buffer fault attribute.

Public Members

uint32_t attribute

Indicates the faulting attributes, when a properly-enabled cache write buffer error interrupt event is detected.

struct *_mcm_buffer_fault_attribute._mcm_buffer_fault_attribut* attribute_memory

struct __mcm__buffer__fault__attribut

#include <fsl_mcm.h>

Public Members

uint32_t busErrorDataAccessType

Indicates the type of cache write buffer access.

uint32_t busErrorPrivilegeLevel

Indicates the privilege level of the cache write buffer access.

uint32_t busErrorSize

Indicates the size of the cache write buffer access.

uint32_t busErrorAccess

Indicates the type of system bus access.

uint32_t busErrorMasterID

Indicates the crossbar switch bus master number of the captured cache write buffer bus error.

uint32_t busErrorOverrun

Indicates if another cache write buffer bus error is detected.

union __mcm__lmem__fault__attribute

#include <fsl_mcm.h> The union of LMEM fault attribute.

Public Members

uint32_t attribute

Indicates the attributes of the LMEM fault detected.

struct *_mcm_lmem_fault_attribute._mcm_lmem_fault_attribut* attribute_memory

struct __mcm__lmem__fault__attribut

#include <fsl_mcm.h>

Public Members

uint32_t parityFaultProtectionSignal

Indicates the features of parity fault protection signal.

uint32_t parityFaultMasterSize
: Indicates the parity fault master size.

uint32_t parityFaultWrite
: Indicates the parity fault is caused by read or write.

uint32_t backdoorAccess
: Indicates the LMEM access fault is initiated by core access or backdoor access.

uint32_t parityFaultSyndrome
: Indicates the parity fault syndrome.

uint32_t overrun
: Indicates the number of faultss.

2.36 PDB: Programmable Delay Block

void PDB_Init(PDB_Type *base, const *pdb_config_t* *config)
: Initializes the PDB module.

  This function initializes the PDB module. The operations included are as follows.

  - Enable the clock for PDB instance.
  - Configure the PDB module.
  - Enable the PDB module.

  Parameters

  - base – PDB peripheral base address.
  - config – Pointer to the configuration structure. See "pdb_config_t".

void PDB_Deinit(PDB_Type *base)
: De-initializes the PDB module.

  Parameters

  - base – PDB peripheral base address.

void PDB_GetDefaultConfig(*pdb_config_t* *config)
: Initializes the PDB user configuration structure.

  This function initializes the user configuration structure to a default value. The default values are as follows.

```
config->loadValueMode = kPDB_LoadValueImmediately;
config->prescalerDivider = kPDB_PrescalerDivider1;
config->dividerMultiplicationFactor = kPDB_DividerMultiplicationFactor1;
config->triggerInputSource = kPDB_TriggerSoftware;
config->enableContinuousMode = false;
```

  Parameters

  - config – Pointer to configuration structure. See "pdb_config_t".

static inline void PDB_Enable(PDB_Type *base, bool enable)

Enables the PDB module.

Parameters

- base – PDB peripheral base address.
- enable – Enable the module or not.

static inline void PDB_DoSoftwareTrigger(PDB_Type *base)

Triggers the PDB counter by software.

Parameters

- base – PDB peripheral base address.

static inline void PDB_DoLoadValues(PDB_Type *base)

Loads the counter values.

This function loads the counter values from the internal buffer. See "pdb_load_value_mode_t" about PDB's load mode.

Parameters

- base – PDB peripheral base address.

static inline void PDB_EnableDMA(PDB_Type *base, bool enable)

Enables the DMA for the PDB module.

Parameters

- base – PDB peripheral base address.
- enable – Enable the feature or not.

static inline void PDB_EnableInterrupts(PDB_Type *base, uint32_t mask)

Enables the interrupts for the PDB module.

Parameters

- base – PDB peripheral base address.
- mask – Mask value for interrupts. See "_pdb_interrupt_enable".

static inline void PDB_DisableInterrupts(PDB_Type *base, uint32_t mask)

Disables the interrupts for the PDB module.

Parameters

- base – PDB peripheral base address.
- mask – Mask value for interrupts. See "_pdb_interrupt_enable".

static inline uint32_t PDB_GetStatusFlags(PDB_Type *base)

Gets the status flags of the PDB module.

Parameters

- base – PDB peripheral base address.

Returns

Mask value for asserted flags. See "_pdb_status_flags".

static inline void PDB_ClearStatusFlags(PDB_Type *base, uint32_t mask)

Clears the status flags of the PDB module.

Parameters

- base – PDB peripheral base address.
- mask – Mask value of flags. See "_pdb_status_flags".

static inline void PDB_SetModulusValue(PDB_Type *base, uint32_t value)

Specifies the counter period.

Parameters

- base – PDB peripheral base address.
- value – Setting value for the modulus. 16-bit is available.

static inline uint32_t PDB_GetCounterValue(PDB_Type *base)

Gets the PDB counter's current value.

Parameters

- base – PDB peripheral base address.

Returns

PDB counter's current value.

static inline void PDB_SetCounterDelayValue(PDB_Type *base, uint32_t value)

Sets the value for the PDB counter delay event.

Parameters

- base – PDB peripheral base address.
- value – Setting value for PDB counter delay event. 16-bit is available.

static inline void PDB_SetADCPreTriggerConfig(PDB_Type *base, *pdb_adc_trigger_channel_t* channel, *pdb_adc_pretrigger_config_t* *config)

Configures the ADC pre-trigger in the PDB module.

Parameters

- base – PDB peripheral base address.
- channel – Channel index for ADC instance.
- config – Pointer to the configuration structure. See "pdb_adc_pretrigger_config_t".

static inline void PDB_SetADCPreTriggerDelayValue(PDB_Type *base, *pdb_adc_trigger_channel_t* channel, *pdb_adc_pretrigger_t* pretriggerNumber, uint32_t value)

Sets the value for the ADC pre-trigger delay event.

This function sets the value for ADC pre-trigger delay event. It specifies the delay value for the channel's corresponding pre-trigger. The pre-trigger asserts when the PDB counter is equal to the set value.

Parameters

- base – PDB peripheral base address.
- channel – Channel index for ADC instance.
- pretriggerNumber – Channel group index for ADC instance.
- value – Setting value for ADC pre-trigger delay event. 16-bit is available.

static inline uint32_t PDB_GetADCPreTriggerStatusFlags(PDB_Type *base, *pdb_adc_trigger_channel_t* channel)

Gets the ADC pre-trigger's status flags.

Parameters

- base – PDB peripheral base address.
- channel – Channel index for ADC instance.

Returns
Mask value for asserted flags. See "_pdb_adc_pretrigger_flags".

static inline void PDB_ClearADCPreTriggerStatusFlags(PDB_Type *base, *pdb_adc_trigger_channel_t* channel, uint32_t mask)

Clears the ADC pre-trigger status flags.

Parameters

- base – PDB peripheral base address.
- channel – Channel index for ADC instance.
- mask – Mask value for flags. See "_pdb_adc_pretrigger_flags".

void PDB_SetDACTriggerConfig(PDB_Type *base, *pdb_dac_trigger_channel_t* channel, *pdb_dac_trigger_config_t* *config)

Configures the DAC trigger in the PDB module.

Parameters

- base – PDB peripheral base address.
- channel – Channel index for DAC instance.
- config – Pointer to the configuration structure. See "pdb_dac_trigger_config_t".

static inline void PDB_SetDACTriggerIntervalValue(PDB_Type *base, *pdb_dac_trigger_channel_t* channel, uint32_t value)

Sets the value for the DAC interval event.

This function sets the value for DAC interval event. DAC interval trigger triggers the DAC module to update the buffer when the DAC interval counter is equal to the set value.

Parameters

- base – PDB peripheral base address.
- channel – Channel index for DAC instance.
- value – Setting value for the DAC interval event.

static inline void PDB_EnablePulseOutTrigger(PDB_Type *base, *pdb_pulse_out_channel_mask_t* channelMask, bool enable)

Enables the pulse out trigger channels.

Parameters

- base – PDB peripheral base address.
- channelMask – Channel mask value for multiple pulse out trigger channel.
- enable – Whether the feature is enabled or not.

static inline void PDB_SetPulseOutTriggerDelayValue(PDB_Type *base, *pdb_pulse_out_trigger_channel_t* channel, uint32_t value1, uint32_t value2)

Sets event values for the pulse out trigger.

This function is used to set event values for the pulse output trigger. These pulse output trigger delay values specify the delay for the PDB Pulse-out. Pulse-out goes high when the PDB counter is equal to the pulse output high value (value1). Pulse-out goes low when the PDB counter is equal to the pulse output low value (value2).

Parameters

- base – PDB peripheral base address.

- channel – Channel index for pulse out trigger channel.
- value1 – Setting value for pulse out high.
- value2 – Setting value for pulse out low.

FSL_PDB_DRIVER_VERSION

PDB driver version 2.0.4.

enum _pdb_status_flags

PDB flags.

Values:

enumerator kPDB_LoadOKFlag

This flag is automatically cleared when the values in buffers are loaded into the internal registers after the LDOK bit is set or the PDBEN is cleared.

enumerator kPDB_DelayEventFlag

PDB timer delay event flag.

enum _pdb_adc_pretrigger_flags

PDB ADC PreTrigger channel flags.

Values:

enumerator kPDB_ADCPreTriggerChannel0Flag

Pre-trigger 0 flag.

enumerator kPDB_ADCPreTriggerChannel1Flag

Pre-trigger 1 flag.

enumerator kPDB_ADCPreTriggerChannel0ErrorFlag

Pre-trigger 0 Error.

enumerator kPDB_ADCPreTriggerChannel1ErrorFlag

Pre-trigger 1 Error.

enum _pdb_interrupt_enable

PDB buffer interrupts.

Values:

enumerator kPDB_SequenceErrorInterruptEnable

PDB sequence error interrupt enable.

enumerator kPDB_DelayInterruptEnable

PDB delay interrupt enable.

enum _pdb_load_value_mode

PDB load value mode.

Selects the mode to load the internal values after doing the load operation (write 1 to PDBx_SC[LDOK]). These values are for the following operations.

- PDB counter (PDBx_MOD, PDBx_IDLY)
- ADC trigger (PDBx_CHnDLYm)
- DAC trigger (PDBx_DACINTx)
- CMP trigger (PDBx_POyDLY)

Values:

enumerator kPDB_LoadValueImmediately
Load immediately after 1 is written to LDOK.

enumerator kPDB_LoadValueOnCounterOverflow
Load when the PDB counter overflows (reaches the MOD register value).

enumerator kPDB_LoadValueOnTriggerInput
Load a trigger input event is detected.

enumerator kPDB_LoadValueOnCounterOverflowOrTriggerInput
Load either when the PDB counter overflows or a trigger input is detected.

enum _pdb_prescaler_divider
Prescaler divider.

Counting uses the peripheral clock divided by multiplication factor selected by times of MULT.

Values:

enumerator kPDB_PrescalerDivider1
Divider x1.

enumerator kPDB_PrescalerDivider2
Divider x2.

enumerator kPDB_PrescalerDivider4
Divider x4.

enumerator kPDB_PrescalerDivider8
Divider x8.

enumerator kPDB_PrescalerDivider16
Divider x16.

enumerator kPDB_PrescalerDivider32
Divider x32.

enumerator kPDB_PrescalerDivider64
Divider x64.

enumerator kPDB_PrescalerDivider128
Divider x128.

enum _pdb_divider_multiplication_factor

Multiplication factor select for prescaler.

Selects the multiplication factor of the prescaler divider for the counter clock.

Values:

enumerator kPDB_DividerMultiplicationFactor1

Multiplication factor is 1.

enumerator kPDB_DividerMultiplicationFactor10

Multiplication factor is 10.

enumerator kPDB_DividerMultiplicationFactor20

Multiplication factor is 20.

enumerator kPDB_DividerMultiplicationFactor40

Multiplication factor is 40.

enum _pdb_trigger_input_source

Trigger input source.

Selects the trigger input source for the PDB. The trigger input source can be internal or external (EXTRG pin), or the software trigger. See chip configuration details for the actual PDB input trigger connections.

Values:

enumerator kPDB_TriggerInput0

Trigger-In 0.

enumerator kPDB_TriggerInput1

Trigger-In 1.

enumerator kPDB_TriggerInput2

Trigger-In 2.

enumerator kPDB_TriggerInput3

Trigger-In 3.

enumerator kPDB_TriggerInput4

Trigger-In 4.

enumerator kPDB_TriggerInput5

Trigger-In 5.

enumerator kPDB_TriggerInput6

Trigger-In 6.

enumerator kPDB_TriggerInput7

Trigger-In 7.

enumerator kPDB_TriggerInput8
Trigger-In 8.

enumerator kPDB_TriggerInput9
Trigger-In 9.

enumerator kPDB_TriggerInput10
Trigger-In 10.

enumerator kPDB_TriggerInput11
Trigger-In 11.

enumerator kPDB_TriggerInput12
Trigger-In 12.

enumerator kPDB_TriggerInput13
Trigger-In 13.

enumerator kPDB_TriggerInput14
Trigger-In 14.

enumerator kPDB_TriggerSoftware
Trigger-In 15, software trigger.

enum _pdb_adc_trigger_channel
List of PDB ADC trigger channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kPDB_ADCTriggerChannel0
PDB ADC trigger channel number 0

enumerator kPDB_ADCTriggerChannel1
PDB ADC trigger channel number 1

enumerator kPDB_ADCTriggerChannel2
PDB ADC trigger channel number 2

enumerator kPDB_ADCTriggerChannel3
PDB ADC trigger channel number 3

enum _pdb_adc_pretrigger
List of PDB ADC pretrigger.

Note: Actual number of available pretrigger channels is SoC dependent

Values:

enumerator kPDB_ADCPreTrigger0
PDB ADC pretrigger number 0

enumerator kPDB_ADCPreTrigger1
PDB ADC pretrigger number 1

enumerator kPDB_ADCPreTrigger2
PDB ADC pretrigger number 2

enumerator kPDB_ADCPreTrigger3
PDB ADC pretrigger number 3

enumerator kPDB_ADCPreTrigger4
PDB ADC pretrigger number 4

enumerator kPDB_ADCPreTrigger5
PDB ADC pretrigger number 5

enumerator kPDB_ADCPreTrigger6
PDB ADC pretrigger number 6

enumerator kPDB_ADCPreTrigger7
PDB ADC pretrigger number 7

enum _pdb_dac_trigger_channel
List of PDB DAC trigger channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kPDB_DACTriggerChannel0
PDB DAC trigger channel number 0

enumerator kPDB_DACTriggerChannel1
PDB DAC trigger channel number 1

enum _pdb_pulse_out_trigger_channel
List of PDB pulse out trigger channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kPDB_PulseOutTriggerChannel0
PDB pulse out trigger channel number 0

enumerator kPDB_PulseOutTriggerChannel1
PDB pulse out trigger channel number 1

enumerator kPDB_PulseOutTriggerChannel2
PDB pulse out trigger channel number 2

enumerator kPDB_PulseOutTriggerChannel3
PDB pulse out trigger channel number 3

enum _pdb_pulse_out_channel_mask
List of PDB pulse out trigger channels mask.

Note: Actual number of available channels mask is SoC dependent

Values:

enumerator kPDB_PulseOutChannel0Mask
PDB pulse out trigger channel number 0 mask

enumerator kPDB_PulseOutChannel1Mask
PDB pulse out trigger channel number 1 mask

enumerator kPDB_PulseOutChannel2Mask
PDB pulse out trigger channel number 2 mask

enumerator kPDB_PulseOutChannel3Mask
PDB pulse out trigger channel number 3 mask

typedef enum *_pdb_load_value_mode* pdb_load_value_mode_t
PDB load value mode.

Selects the mode to load the internal values after doing the load operation (write 1 to PDBx_SC[LDOK]). These values are for the following operations.

- PDB counter (PDBx_MOD, PDBx_IDLY)
- ADC trigger (PDBx_CHnDLYm)
- DAC trigger (PDBx_DACINTx)
- CMP trigger (PDBx_POyDLY)

typedef enum *_pdb_prescaler_divider* pdb_prescaler_divider_t
Prescaler divider.

Counting uses the peripheral clock divided by multiplication factor selected by times of MULT.

typedef enum *_pdb_divider_multiplication_factor* pdb_divider_multiplication_factor_t
Multiplication factor select for prescaler.

Selects the multiplication factor of the prescaler divider for the counter clock.

typedef enum *_pdb_trigger_input_source* pdb_trigger_input_source_t

Trigger input source.

Selects the trigger input source for the PDB. The trigger input source can be internal or external (EXTRG pin), or the software trigger. See chip configuration details for the actual PDB input trigger connections.

typedef enum *_pdb_adc_trigger_channel* pdb_adc_trigger_channel_t

List of PDB ADC trigger channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_pdb_adc_pretrigger* pdb_adc_pretrigger_t

List of PDB ADC pretrigger.

Note: Actual number of available pretrigger channels is SoC dependent

typedef enum *_pdb_dac_trigger_channel* pdb_dac_trigger_channel_t

List of PDB DAC trigger channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_pdb_pulse_out_trigger_channel* pdb_pulse_out_trigger_channel_t

List of PDB pulse out trigger channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_pdb_pulse_out_channel_mask* pdb_pulse_out_channel_mask_t

List of PDB pulse out trigger channels mask.

Note: Actual number of available channels mask is SoC dependent

typedef struct *_pdb_config* pdb_config_t

PDB module configuration.

typedef struct *_pdb_adc_pretrigger_config* pdb_adc_pretrigger_config_t

PDB ADC Pre-trigger configuration.

typedef struct *_pdb_dac_trigger_config* pdb_dac_trigger_config_t

PDB DAC trigger configuration.

struct _pdb_config

#include <fsl_pdb.h> PDB module configuration.

Public Members

pdb_load_value_mode_t loadValueMode
Select the load value mode.

pdb_prescaler_divider_t prescalerDivider
Select the prescaler divider.

pdb_divider_multiplication_factor_t dividerMultiplicationFactor
Multiplication factor select for prescaler.

pdb_trigger_input_source_t triggerInputSource
Select the trigger input source.

bool enableContinuousMode
Enable the PDB operation in Continuous mode.

struct __pdb_adc_pretrigger_config
#include <fsl_pdb.h> PDB ADC Pre-trigger configuration.

Public Members

uint32_t enablePreTriggerMask
PDB Channel Pre-trigger Enable.

uint32_t enableOutputMask
PDB Channel Pre-trigger Output Select. PDB channel's corresponding pre-trigger asserts when the counter reaches the channel delay register.

uint32_t enableBackToBackOperationMask
PDB Channel pre-trigger Back-to-Back Operation Enable. Back-to-back operation enables the ADC conversions complete to trigger the next PDB channel pre-trigger and trigger output, so that the ADC conversions can be triggered on next set of configuration and results registers.

struct __pdb_dac_trigger_config
#include <fsl_pdb.h> PDB DAC trigger configuration.

Public Members

bool enableExternalTriggerInput
Enables the external trigger for DAC interval counter.

bool enableIntervalTrigger
Enables the DAC interval trigger.

2.37 PIT: Periodic Interrupt Timer

void PIT_Init(PIT_Type *base, const *pit_config_t* *config)

Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.

Note: This API should be called at the beginning of the application using the PIT driver.

Parameters

- base – PIT peripheral base address
- config – Pointer to the user's PIT config structure

void PIT_Deinit(PIT_Type *base)

Gates the PIT clock and disables the PIT module.

Parameters

- base – PIT peripheral base address

static inline void PIT_GetDefaultConfig(*pit_config_t* *config)

Fills in the PIT configuration structure with the default settings.

The default values are as follows.

```
config->enableRunInDebug = false;
```

Parameters

- config – Pointer to the configuration structure.

static inline void PIT_SetTimerChainMode(PIT_Type *base, *pit_chnl_t* channel, bool enable)

Enables or disables chaining a timer with the previous timer.

When a timer has a chain mode enabled, it only counts after the previous timer has expired. If the timer n-1 has counted down to 0, counter n decrements the value by one. Each timer is 32-bits, which allows the developers to chain timers together and form a longer timer (64-bits and larger). The first timer (timer 0) can't be chained to any other timer.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number which is chained with the previous timer
- enable – Enable or disable chain. true: Current timer is chained with the previous timer. false: Timer doesn't chain with other timers.

static inline void PIT_EnableInterrupts(PIT_Type *base, *pit_chnl_t* channel, uint32_t mask)

Enables the selected PIT interrupts.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number
- mask – The interrupts to enable. This is a logical OR of members of the enumeration *pit_interrupt_enable_t*

static inline void PIT_DisableInterrupts(PIT_Type *base, *pit_chnl_t* channel, uint32_t mask)

Disables the selected PIT interrupts.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number
- mask – The interrupts to disable. This is a logical OR of members of the enumeration *pit_interrupt_enable_t*

static inline uint32_t PIT_GetEnabledInterrupts(PIT_Type *base, *pit_chnl_t* channel)

Gets the enabled PIT interrupts.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *pit_interrupt_enable_t*

static inline uint32_t PIT_GetStatusFlags(PIT_Type *base, *pit_chnl_t* channel)

Gets the PIT status flags.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number

Returns

The status flags. This is the logical OR of members of the enumeration *pit_status_flags_t*

static inline void PIT_ClearStatusFlags(PIT_Type *base, *pit_chnl_t* channel, uint32_t mask)

Clears the PIT status flags.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number
- mask – The status flags to clear. This is a logical OR of members of the enumeration *pit_status_flags_t*

static inline void PIT_SetTimerPeriod(PIT_Type *base, *pit_chnl_t* channel, uint32_t count)

Sets the timer period in units of count.

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note: Users can call the utility macros provided in fsl_common.h to convert to ticks.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number
- count – Timer period in units of ticks

static inline uint32_t PIT_GetCurrentTimerCount(PIT_Type *base, *pit_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: Users can call the utility macros provided in fsl_common.h to convert ticks to usec or msec.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number

Returns

Current timer counting value in ticks

static inline void PIT_StartTimer(PIT_Type *base, *pit_chnl_t* channel)

Starts the timer counting.

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number.

static inline void PIT_StopTimer(PIT_Type *base, *pit_chnl_t* channel)

Stops the timer counting.

This function stops every timer counting. Timers reload their periods respectively after the next time they call the PIT_DRV_StartTimer.

Parameters

- base – PIT peripheral base address
- channel – Timer channel number.

FSL_PIT_DRIVER_VERSION

PIT Driver Version 2.2.0.

enum _pit_chnl

List of PIT channels.

Note: Actual number of available channels is SoC dependent

Values:

enumerator kPIT_Chnl_0

PIT channel number 0

enumerator kPIT_Chnl_1

PIT channel number 1

enumerator kPIT_Chnl_2
PIT channel number 2

enumerator kPIT_Chnl_3
PIT channel number 3

enum _pit_interrupt_enable
List of PIT interrupts.

Values:

enumerator kPIT_TimerInterruptEnable
Timer interrupt enable

enum _pit_status_flags
List of PIT status flags.

Values:

enumerator kPIT_TimerFlag
Timer flag

typedef enum *_pit_chnl* pit_chnl_t
List of PIT channels.

Note: Actual number of available channels is SoC dependent

typedef enum *_pit_interrupt_enable* pit_interrupt_enable_t
List of PIT interrupts.

typedef enum *_pit_status_flags* pit_status_flags_t
List of PIT status flags.

typedef struct *_pit_config* pit_config_t
PIT configuration structure.

This structure holds the configuration settings for the PIT peripheral. To initialize this structure to reasonable defaults, call the *PIT_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The configuration structure can be made constant so it resides in flash.

uint64_t PIT_GetLifetimeTimerCount(PIT_Type *base)
Reads the current lifetime counter value.

The lifetime timer is a 64-bit timer which chains timer 0 and timer 1 together. Timer 0 and 1 are chained by calling the PIT_SetTimerChainMode before using this timer. The period of lifetime timer is equal to the “period of timer 0 * period of timer 1”. For the 64-bit value, the higher 32-bit has the value of timer 1, and the lower 32-bit has the value of timer 0.

Parameters

- base – PIT peripheral base address

Returns
Current lifetime timer value

struct _pit_config

#include <fsl_pit.h> PIT configuration structure.

This structure holds the configuration settings for the PIT peripheral. To initialize this structure to reasonable defaults, call the *PIT_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The configuration structure can be made constant so it resides in flash.

Public Members

bool enableRunInDebug

true: Timers run in debug mode; false: Timers stop in debug mode

2.38 PMC: Power Management Controller

static inline void PMC_GetVersionId(PMC_Type *base, *pmc_version_id_t* *versionId)

Gets the PMC version ID.

This function gets the PMC version ID, including major version number, minor version number, and a feature specification number.

Parameters

- base – PMC peripheral base address.
- versionId – Pointer to version ID structure.

void PMC_GetParam(PMC_Type *base, *pmc_param_t* *param)

Gets the PMC parameter.

This function gets the PMC parameter including the VLPO enable and the HVD enable.

Parameters

- base – PMC peripheral base address.
- param – Pointer to PMC param structure.

void PMC_ConfigureLowVoltDetect(PMC_Type *base, const *pmc_low_volt_detect_config_t* *config)

Configures the low-voltage detect setting.

This function configures the low-voltage detect setting, including the trip point voltage setting, enables or disables the interrupt, enables or disables the system reset.

Parameters

- base – PMC peripheral base address.
- config – Low-voltage detect configuration structure.

static inline bool PMC_GetLowVoltDetectFlag(PMC_Type *base)

Gets the Low-voltage Detect Flag status.

This function reads the current LVDF status. If it returns 1, a low-voltage event is detected.

Parameters

- base – PMC peripheral base address.

Returns
Current low-voltage detect flag

- true: Low-voltage detected
- false: Low-voltage not detected

static inline void PMC_ClearLowVoltDetectFlag(PMC_Type *base)

Acknowledges clearing the Low-voltage Detect flag.

This function acknowledges the low-voltage detection errors (write 1 to clear LVDF).

Parameters

- base – PMC peripheral base address.

void PMC_ConfigureLowVoltWarning(PMC_Type *base, const *pmc_low_volt_warning_config_t* *config)

Configures the low-voltage warning setting.

This function configures the low-voltage warning setting, including the trip point voltage setting and enabling or disabling the interrupt.

Parameters

- base – PMC peripheral base address.
- config – Low-voltage warning configuration structure.

static inline bool PMC_GetLowVoltWarningFlag(PMC_Type *base)

Gets the Low-voltage Warning Flag status.

This function polls the current LVWF status. When 1 is returned, it indicates a low-voltage warning event. LVWF is set when V Supply transitions below the trip point or after reset and V Supply is already below the V LVW.

Parameters

- base – PMC peripheral base address.

Returns
Current LVWF status

- true: Low-voltage Warning Flag is set.
- false: the Low-voltage Warning does not happen.

static inline void PMC_ClearLowVoltWarningFlag(PMC_Type *base)

Acknowledges the Low-voltage Warning flag.

This function acknowledges the low voltage warning errors (write 1 to clear LVWF).

Parameters

- base – PMC peripheral base address.

void PMC_ConfigureHighVoltDetect(PMC_Type *base, const *pmc_high_volt_detect_config_t* *config)

Configures the high-voltage detect setting.

This function configures the high-voltage detect setting, including the trip point voltage setting, enabling or disabling the interrupt, enabling or disabling the system reset.

Parameters

- base – PMC peripheral base address.
- config – High-voltage detect configuration structure.

static inline bool PMC_GetHighVoltDetectFlag(PMC_Type *base)

Gets the High-voltage Detect Flag status.

This function reads the current HVDF status. If it returns 1, a low voltage event is detected.

Parameters

- base – PMC peripheral base address.

Returns

Current high-voltage detect flag

- true: High-voltage detected
- false: High-voltage not detected

static inline void PMC_ClearHighVoltDetectFlag(PMC_Type *base)

Acknowledges clearing the High-voltage Detect flag.

This function acknowledges the high-voltage detection errors (write 1 to clear HVDF).

Parameters

- base – PMC peripheral base address.

void PMC_ConfigureBandgapBuffer(PMC_Type *base, const *pmc_bandgap_buffer_config_t* *config)

Configures the PMC bandgap.

This function configures the PMC bandgap, including the drive select and behavior in low-power mode.

Parameters

- base – PMC peripheral base address.
- config – Pointer to the configuration structure

static inline bool PMC_GetPeriphIOIsolationFlag(PMC_Type *base)

Gets the acknowledge Peripherals and I/O pads isolation flag.

This function reads the Acknowledge Isolation setting that indicates whether certain peripherals and the I/O pads are in a latched state as a result of having been in the VLLS mode.

Parameters

- base – PMC peripheral base address.
- base – Base address for current PMC instance.

Returns

ACK isolation 0 - Peripherals and I/O pads are in a normal run state. 1 - Certain peripherals and I/O pads are in an isolated and latched state.

static inline void PMC_ClearPeriphIOIsolationFlag(PMC_Type *base)

Acknowledges the isolation flag to Peripherals and I/O pads.

This function clears the ACK Isolation flag. Writing one to this setting when it is set releases the I/O pads and certain peripherals to their normal run mode state.

Parameters

- base – PMC peripheral base address.

static inline bool PMC_IsRegulatorInRunRegulation(PMC_Type *base)

Gets the regulator regulation status.

This function returns the regulator to run a regulation status. It provides the current status of the internal voltage regulator.

Parameters

- base – PMC peripheral base address.
- base – Base address for current PMC instance.

Returns

Regulation status 0 - Regulator is in a stop regulation or in transition to/from the regulation. 1 - Regulator is in a run regulation.

FSL_PMC_DRIVER_VERSION

PMC driver version.

Version 2.0.4.

enum _pmc_low_volt_detect_volt_select

Low-voltage Detect Voltage Select.

Values:

enumerator kPMC_LowVoltDetectLowTrip

Low-trip point selected (VLVD = VLVDL)

enumerator kPMC_LowVoltDetectHighTrip

High-trip point selected (VLVD = VLVDH)

enum _pmc_low_volt_warning_volt_select

Low-voltage Warning Voltage Select.

Values:

enumerator kPMC_LowVoltWarningLowTrip

Low-trip point selected (VLVW = VLVW1)

enumerator kPMC_LowVoltWarningMid1Trip

Mid 1 trip point selected (VLVW = VLVW2)

enumerator kPMC_LowVoltWarningMid2Trip

Mid 2 trip point selected (VLVW = VLVW3)

enumerator kPMC_LowVoltWarningHighTrip

High-trip point selected (VLVW = VLVW4)

enum _pmc_high_volt_detect_volt_select

High-voltage Detect Voltage Select.

Values:

enumerator kPMC_HighVoltDetectLowTrip

Low-trip point selected (VHVD = VHVDL)

enumerator kPMC_HighVoltDetectHighTrip

High-trip point selected (VHVD = VHVDH)

enum __pmc__bandgap__buffer__drive__select

Bandgap Buffer Drive Select.

Values:

enumerator kPMC__BandgapBufferDriveLow

Low-drive.

enumerator kPMC__BandgapBufferDriveHigh

High-drive.

enum __pmc__vlp__freq__option

VLPx Option.

Values:

enumerator kPMC__FreqRestrict

Frequency is restricted in VLPx mode.

enumerator kPMC__FreqUnrestrict

Frequency is unrestricted in VLPx mode.

typedef enum *_pmc_low_volt_detect_volt_select* pmc__low__volt__detect__volt__select__t

Low-voltage Detect Voltage Select.

typedef enum *_pmc_low_volt_warning_volt_select* pmc__low__volt__warning__volt__select__t

Low-voltage Warning Voltage Select.

typedef enum *_pmc_high_volt_detect_volt_select* pmc__high__volt__detect__volt__select__t

High-voltage Detect Voltage Select.

typedef enum *_pmc_bandgap_buffer_drive_select* pmc__bandgap__buffer__drive__select__t

Bandgap Buffer Drive Select.

typedef enum *_pmc_vlp_freq_option* pmc__vlp__freq__mode__t

VLPx Option.

typedef struct *_pmc_version_id* pmc__version__id__t

IP version ID definition.

typedef struct *_pmc_param* pmc__param__t

IP parameter definition.

typedef struct *_pmc_low_volt_detect_config* pmc__low__volt__detect__config__t

Low-voltage Detect Configuration Structure.

typedef struct *_pmc_low_volt_warning_config* pmc__low__volt__warning__config__t

Low-voltage Warning Configuration Structure.

typedef struct *_pmc_high_volt_detect_config* pmc_high_volt_detect_config_t
High-voltage Detect Configuration Structure.

typedef struct *_pmc_bandgap_buffer_config* pmc_bandgap_buffer_config_t
Bandgap Buffer configuration.

struct __pmc_version_id
#include <fsl_pmc.h> IP version ID definition.

Public Members

uint16_t feature
Feature Specification Number.

uint8_t minor
Minor version number.

uint8_t major
Major version number.

struct __pmc_param
#include <fsl_pmc.h> IP parameter definition.

Public Members

bool vlpoEnable
VLPO enable.

bool hvdEnable
HVD enable.

struct __pmc_low_volt_detect_config
#include <fsl_pmc.h> Low-voltage Detect Configuration Structure.

Public Members

bool enableInt
Enable interrupt when Low-voltage detect

bool enableReset
Enable system reset when Low-voltage detect

pmc_low_volt_detect_volt_select_t voltSelect
Low-voltage detect trip point voltage selection

struct __pmc_low_volt_warning_config
#include <fsl_pmc.h> Low-voltage Warning Configuration Structure.

Public Members

bool enableInt
Enable interrupt when low-voltage warning

pmc_low_volt_warning_volt_select_t voltSelect
Low-voltage warning trip point voltage selection

struct __pmc_high_volt_detect_config
#include <fsl_pmc.h> High-voltage Detect Configuration Structure.

Public Members

bool enableInt
Enable interrupt when high-voltage detect

bool enableReset
Enable system reset when high-voltage detect

pmc_high_volt_detect_volt_select_t voltSelect
High-voltage detect trip point voltage selection

struct __pmc_bandgap_buffer_config
#include <fsl_pmc.h> Bandgap Buffer configuration.

Public Members

bool enable
Enable bandgap buffer.

bool enableInLowPowerMode
Enable bandgap buffer in low-power mode.

pmc_bandgap_buffer_drive_select_t drive
Bandgap buffer drive select.

2.39 PORT: Port Control and Interrupts

static inline void PORT_SetPinConfig(PORT_Type *base, uint32_t pin, const *port_pin_config_t* *config)
Sets the port PCR register.
This is an example to define an input pin or output pin PCR configuration.

```
// Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnLockRegister,
};
```

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- config – PORT PCR register configuration structure.

static inline void PORT_SetMultiplePinsConfig(PORT_Type *base, uint32_t mask, const *port_pin_config_t* *config)

Sets the port PCR register for multiple pins.

This is an example to define input pins or output pins PCR configuration.

```
Define a digital input pin PCR configuration
port_pin_config_t config = {
    kPORT_PullUp ,
    kPORT_PullEnable,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnlockRegister,
};
```

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT PCR register configuration structure.

static inline void PORT_SetMultipleInterruptPinsConfig(PORT_Type *base, uint32_t mask, *port_interrupt_t* config)

Sets the port interrupt configuration in PCR register for multiple pins.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- config – PORT pin interrupt configuration.
  - *kPORT_InterruptOrDMADisabled*: Interrupt/DMA request disabled.
  - *kPORT_DMARisingEdge* : DMA request on rising edge(if the DMA requests exit).
  - *kPORT_DMAFallingEdge*: DMA request on falling edge(if the DMA requests exit).

- *kPORT_DMAEitherEdge* : DMA request on either edge(if the DMA requests exit).
- *kPORT_FlagRisingEdge* : Flag sets on rising edge(if the Flag states exit).
- *kPORT_FlagFallingEdge* : Flag sets on falling edge(if the Flag states exit).
- *kPORT_FlagEitherEdge* : Flag sets on either edge(if the Flag states exit).
- *kPORT_InterruptLogicZero* : Interrupt when logic zero.
- *kPORT_InterruptRisingEdge* : Interrupt on rising edge.
- *kPORT_InterruptFallingEdge*: Interrupt on falling edge.
- *kPORT_InterruptEitherEdge* : Interrupt on either edge.
- *kPORT_InterruptLogicOne* : Interrupt when logic one.
- *kPORT_ActiveHighTriggerOutputEnable* : Enable active high-trigger output (if the trigger states exit).
- *kPORT_ActiveLowTriggerOutputEnable* : Enable active low-trigger output (if the trigger states exit).

static inline void PORT_SetPinMux(PORT_Type *base, uint32_t pin, *port_mux_t* mux)

Configures the pin muxing.

Note: : This function is NOT recommended to use together with the PORT_SetPinsConfig, because the PORT_SetPinsConfig need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : kPORT_PinDisabledOrAnalog). This function is recommended to use to reset the pin mux

---

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- mux – pin muxing slot selection.
  - *kPORT_PinDisabledOrAnalog*: Pin disabled or work in analog function.
  - *kPORT_MuxAsGpio* : Set as GPIO.
  - *kPORT_MuxAlt2* : chip-specific.
  - *kPORT_MuxAlt3* : chip-specific.
  - *kPORT_MuxAlt4* : chip-specific.
  - *kPORT_MuxAlt5* : chip-specific.
  - *kPORT_MuxAlt6* : chip-specific.
  - *kPORT_MuxAlt7* : chip-specific.

static inline void PORT_EnablePinsDigitalFilter(PORT_Type *base, uint32_t mask, bool enable)

Enables the digital filter in one port, each bit of the 32-bit register represents one pin.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.
- enable – PORT digital filter configuration.

static inline void PORT_SetDigitalFilterConfig(PORT_Type *base, const *port_digital_filter_config_t* *config)

Sets the digital filter in one port, each bit of the 32-bit register represents one pin.

Parameters

- base – PORT peripheral base pointer.
- config – PORT digital filter configuration structure.

static inline void PORT_SetPinInterruptConfig(PORT_Type *base, uint32_t pin, *port_interrupt_t* config)

Configures the port pin interrupt/DMA request.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- config – PORT pin interrupt configuration.
  - *kPORT_InterruptOrDMADisabled*: Interrupt/DMA request disabled.
  - *kPORT_DMARisingEdge* : DMA request on rising edge(if the DMA requests exit).
  - *kPORT_DMAFallingEdge*: DMA request on falling edge(if the DMA requests exit).
  - *kPORT_DMAEitherEdge* : DMA request on either edge(if the DMA requests exit).
  - *kPORT_FlagRisingEdge* : Flag sets on rising edge(if the Flag states exit).
  - *kPORT_FlagFallingEdge* : Flag sets on falling edge(if the Flag states exit).
  - *kPORT_FlagEitherEdge* : Flag sets on either edge(if the Flag states exit).
  - *kPORT_InterruptLogicZero* : Interrupt when logic zero.
  - *kPORT_InterruptRisingEdge* : Interrupt on rising edge.
  - *kPORT_InterruptFallingEdge*: Interrupt on falling edge.
  - *kPORT_InterruptEitherEdge* : Interrupt on either edge.
  - *kPORT_InterruptLogicOne* : Interrupt when logic one.
  - *kPORT_ActiveHighTriggerOutputEnable* : Enable active high-trigger output (if the trigger states exit).
  - *kPORT_ActiveLowTriggerOutputEnable* : Enable active low-trigger output (if the trigger states exit).

static inline void PORT_SetPinDriveStrength(PORT_Type *base, uint32_t pin, uint8_t strength)

Configures the port pin drive strength.

Parameters

- base – PORT peripheral base pointer.
- pin – PORT pin number.
- strength – PORT pin drive strength
  - *kPORT_LowDriveStrength* = 0U - Low-drive strength is configured.
  - *kPORT_HighDriveStrength* = 1U - High-drive strength is configured.

static inline uint32_t PORT_GetPinsInterruptFlags(PORT_Type *base)

Reads the whole port status flag.

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

- base – PORT peripheral base pointer.

Returns

Current port interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

static inline void PORT_ClearPinsInterruptFlags(PORT_Type *base, uint32_t mask)

Clears the multiple pin interrupt status flag.

Parameters

- base – PORT peripheral base pointer.
- mask – PORT pin number macro.

FSL_PORT_DRIVER_VERSION

PORT driver version.

enum _port_pull

Internal resistor pull feature selection.

Values:

enumerator kPORT_PullDisable

Internal pull-up/down resistor is disabled.

enumerator kPORT_PullDown

Internal pull-down resistor is enabled.

enumerator kPORT_PullUp

Internal pull-up resistor is enabled.

enum _port_slew_rate

Slew rate selection.

Values:

enumerator kPORT_FastSlewRate

Fast slew rate is configured.

enumerator kPORT_SlowSlewRate

Slow slew rate is configured.

enum _port_open_drain_enable

Open Drain feature enable/disable.

Values:

enumerator kPORT_OpenDrainDisable
Open drain output is disabled.

enumerator kPORT_OpenDrainEnable
Open drain output is enabled.

enum _port_passive_filter_enable
Passive filter feature enable/disable.
Values:

enumerator kPORT_PassiveFilterDisable
Passive input filter is disabled.

enumerator kPORT_PassiveFilterEnable
Passive input filter is enabled.

enum _port_drive_strength
Configures the drive strength.
Values:

enumerator kPORT_LowDriveStrength
Low-drive strength is configured.

enumerator kPORT_HighDriveStrength
High-drive strength is configured.

enum _port_lock_register
Unlock/lock the pin control register field[15:0].
Values:

enumerator kPORT_UnlockRegister
Pin Control Register fields [15:0] are not locked.

enumerator kPORT_LockRegister
Pin Control Register fields [15:0] are locked.

enum _port_mux
Pin mux selection.
Values:

enumerator kPORT_PinDisabledOrAnalog
Corresponding pin is disabled, but is used as an analog pin.

enumerator kPORT_MuxAsGpio
Corresponding pin is configured as GPIO.

enumerator kPORT_MuxAlt0
Chip-specific

enumerator kPORT_MuxAlt1
Chip-specific

enumerator kPORT_MuxAlt2
Chip-specific

enumerator kPORT_MuxAlt3
Chip-specific

enumerator kPORT_MuxAlt4
Chip-specific

enumerator kPORT_MuxAlt5
Chip-specific

enumerator kPORT_MuxAlt6
Chip-specific

enumerator kPORT_MuxAlt7
Chip-specific

enumerator kPORT_MuxAlt8
Chip-specific

enumerator kPORT_MuxAlt9
Chip-specific

enumerator kPORT_MuxAlt10
Chip-specific

enumerator kPORT_MuxAlt11
Chip-specific

enumerator kPORT_MuxAlt12
Chip-specific

enumerator kPORT_MuxAlt13
Chip-specific

enumerator kPORT_MuxAlt14
Chip-specific

enumerator kPORT_MuxAlt15
Chip-specific

enum _port_interrupt
Configures the interrupt generation condition.

Values:

enumerator kPORT_InterruptOrDMADisabled
Interrupt/DMA request is disabled.

enumerator kPORT_DMARisingEdge
DMA request on rising edge.

enumerator kPORT_DMAFallingEdge
DMA request on falling edge.

enumerator kPORT_DMAEitherEdge
DMA request on either edge.

enumerator kPORT_FlagRisingEdge
Flag sets on rising edge.

enumerator kPORT_FlagFallingEdge
Flag sets on falling edge.

enumerator kPORT_FlagEitherEdge
Flag sets on either edge.

enumerator kPORT_InterruptLogicZero
Interrupt when logic zero.

enumerator kPORT_InterruptRisingEdge
Interrupt on rising edge.

enumerator kPORT_InterruptFallingEdge
Interrupt on falling edge.

enumerator kPORT_InterruptEitherEdge
Interrupt on either edge.

enumerator kPORT_InterruptLogicOne
Interrupt when logic one.

enumerator kPORT_ActiveHighTriggerOutputEnable
Enable active high-trigger output.

enumerator kPORT_ActiveLowTriggerOutputEnable
Enable active low-trigger output.

enum _port_digital_filter_clock_source
Digital filter clock source selection.
Values:

enumerator kPORT_BusClock
Digital filters are clocked by the bus clock.

enumerator kPORT_LpoClock
Digital filters are clocked by the 1 kHz LPO clock.

typedef enum *_port_mux* port_mux_t
Pin mux selection.

typedef enum *_port_interrupt* port_interrupt_t
Configures the interrupt generation condition.

typedef enum *_port_digital_filter_clock_source* port_digital_filter_clock_source_t
Digital filter clock source selection.

typedef struct *_port_digital_filter_config* port_digital_filter_config_t
PORT digital filter feature configuration definition.

typedef struct *_port_pin_config* port_pin_config_t
PORT pin configuration structure.

FSL_COMPONENT_ID

struct _port_digital_filter_config
#include <fsl_port.h> PORT digital filter feature configuration definition.

Public Members

uint32_t digitalFilterWidth
Set digital filter width

port_digital_filter_clock_source_t clockSource
Set digital filter clockSource

struct _port_pin_config
#include <fsl_port.h> PORT pin configuration structure.

Public Members

uint16_t pullSelect
No-pull/pull-down/pull-up select

uint16_t slewRate
Fast/slow slew rate Configure

uint16_t passiveFilterEnable
Passive filter enable/disable

uint16_t openDrainEnable
: Open drain enable/disable

uint16_t driveStrength
: Fast/slow drive strength configure

uint16_t lockRegister
: Lock/unlock the PCR field[15:0]

2.40 RCM: Reset Control Module Driver

static inline void RCM_GetVersionId(RCM_Type *base, *rcm_version_id_t* *versionId)

Gets the RCM version ID.

This function gets the RCM version ID including the major version number, the minor version number, and the feature specification number.

Parameters

- base – RCM peripheral base address.
- versionId – Pointer to the version ID structure.

static inline uint32_t RCM_GetResetSourceImplementedStatus(RCM_Type *base)

Gets the reset source implemented status.

This function gets the RCM parameter that indicates whether the corresponding reset source is implemented. Use source masks defined in the rcm_reset_source_t to get the desired source status.

This is an example.

```
uint32_t status;

To test whether the MCU is reset using Watchdog.
status = RCM_GetResetSourceImplementedStatus(RCM) & (kRCM_SourceWdog | kRCM_SourcePin);
```

Parameters

- base – RCM peripheral base address.

Returns

All reset source implemented status bit map.

static inline uint32_t RCM_GetPreviousResetSources(RCM_Type *base)

Gets the reset source status which caused a previous reset.

This function gets the current reset source status. Use source masks defined in the rcm_reset_source_t to get the desired source status.

This is an example.

```
uint32_t resetStatus;

To get all reset source statuses.
resetStatus = RCM_GetPreviousResetSources(RCM) & kRCM_SourceAll;

To test whether the MCU is reset using Watchdog.
resetStatus = RCM_GetPreviousResetSources(RCM) & kRCM_SourceWdog;
```

(continues on next page)

(continued from previous page)

```
To test multiple reset sources.
resetStatus = RCM_GetPreviousResetSources(RCM) & (kRCM_SourceWdog | kRCM_SourcePin);
```

Parameters

- base – RCM peripheral base address.

Returns

All reset source status bit map.

static inline uint32_t RCM_GetStickyResetSources(RCM_Type *base)

Gets the sticky reset source status.

This function gets the current reset source status that has not been cleared by software for a specific source.

This is an example.

```
uint32_t resetStatus;

To get all reset source statuses.
resetStatus = RCM_GetStickyResetSources(RCM) & kRCM_SourceAll;

To test whether the MCU is reset using Watchdog.
resetStatus = RCM_GetStickyResetSources(RCM) & kRCM_SourceWdog;

To test multiple reset sources.
resetStatus = RCM_GetStickyResetSources(RCM) & (kRCM_SourceWdog | kRCM_SourcePin);
```

Parameters

- base – RCM peripheral base address.

Returns

All reset source status bit map.

static inline void RCM_ClearStickyResetSources(RCM_Type *base, uint32_t sourceMasks)

Clears the sticky reset source status.

This function clears the sticky system reset flags indicated by source masks.

This is an example.

```
Clears multiple reset sources.
RCM_ClearStickyResetSources(kRCM_SourceWdog | kRCM_SourcePin);
```

Parameters

- base – RCM peripheral base address.
- sourceMasks – reset source status bit map

void RCM_ConfigureResetPinFilter(RCM_Type *base, const *rcm_reset_pin_filter_config_t* *config)

Configures the reset pin filter.

This function sets the reset pin filter including the filter source, filter width, and so on.

Parameters

- base – RCM peripheral base address.
- config – Pointer to the configuration structure.

static inline bool RCM_GetEasyPortModePinStatus(RCM_Type *base)

Gets the EZP_MS_B pin assert status.

This function gets the easy port mode status (EZP_MS_B) pin assert status.

Parameters

- base – RCM peripheral base address.

Returns

status true - asserted, false - reasserted

static inline *rcm_boot_rom_config_t* RCM_GetBootRomSource(RCM_Type *base)

Gets the ROM boot source.

This function gets the ROM boot source during the last chip reset.

Parameters

- base – RCM peripheral base address.

Returns

The ROM boot source.

static inline void RCM_ClearBootRomSource(RCM_Type *base)

Clears the ROM boot source flag.

This function clears the ROM boot source flag.

Parameters

- base – Register base address of RCM

void RCM_SetForceBootRomSource(RCM_Type *base, *rcm_boot_rom_config_t* config)

Forces the boot from ROM.

This function forces booting from ROM during all subsequent system resets.

Parameters

- base – RCM peripheral base address.
- config – Boot configuration.

static inline void RCM_SetSystemResetInterruptConfig(RCM_Type *base, uint32_t intMask, *rcm_reset_delay_t* delay)

Sets the system reset interrupt configuration.

For a graceful shut down, the RCM supports delaying the assertion of the system reset for a period of time when the reset interrupt is generated. This function can be used to enable the interrupt and the delay period. The interrupts are passed in as bit mask. See rcm_int_t for details. For example, to delay a reset for 512 LPO cycles after the WDOG timeout or loss-of-clock occurs, configure as follows: RCM_SetSystemResetInterruptConfig(kRCM_IntWatchDog | kRCM_IntLossOfClk, kRCM_ResetDelay512Lpo);

Parameters

- base – RCM peripheral base address.
- intMask – Bit mask of the system reset interrupts to enable. See rcm_interrupt_enable_t for details.
- delay – Bit mask of the system reset interrupts to enable.

FSL_RCM_DRIVER_VERSION

RCM driver version 2.0.6.

enum _rcm_reset_source

System Reset Source Name definitions.

Values:

enumerator kRCM_SourceWakeup

Low-leakage wakeup reset

enumerator kRCM_SourceLvd

Low-voltage detect reset

enumerator kRCM_SourceLoc

Loss of clock reset

enumerator kRCM_SourceLol

Loss of lock reset

enumerator kRCM_SourceWdog

Watchdog reset

enumerator kRCM_SourcePin

External pin reset

enumerator kRCM_SourcePor

Power on reset

enumerator kRCM_SourceJtag

JTAG generated reset

enumerator kRCM_SourceLockup

Core lock up reset

enumerator kRCM_SourceSw

Software reset

enumerator kRCM_SourceMdmap

MDM-AP system reset

enumerator kRCM_SourceEzpt

EzPort reset

enumerator kRCM_SourceSackerr

Parameter could get all reset flags

enumerator kRCM_SourceAll

enum _rcm_run_wait_filter_mode

Reset pin filter select in Run and Wait modes.

Values:

enumerator kRCM_FilterDisable
All filtering disabled

enumerator kRCM_FilterBusClock
Bus clock filter enabled

enumerator kRCM_FilterLpoClock
LPO clock filter enabled

enum _rcm_boot_rom_config
Boot from ROM configuration.
Values:

enumerator kRCM_BootFlash
Boot from flash

enumerator kRCM_BootRomCfg0
Boot from boot ROM due to BOOTCFG0

enumerator kRCM_BootRomFopt
Boot from boot ROM due to FOPT[7]

enumerator kRCM_BootRomBoth
Boot from boot ROM due to both BOOTCFG0 and FOPT[7]

enum _rcm_reset_delay
Maximum delay time from interrupt asserts to system reset.
Values:

enumerator kRCM_ResetDelay8Lpo
Delay 8 LPO cycles.

enumerator kRCM_ResetDelay32Lpo
Delay 32 LPO cycles.

enumerator kRCM_ResetDelay128Lpo
Delay 128 LPO cycles.

enumerator kRCM_ResetDelay512Lpo
Delay 512 LPO cycles.

enum _rcm_interrupt_enable
System reset interrupt enable bit definitions.
Values:

enumerator kRCM_IntNone
No interrupt enabled.

enumerator kRCM_IntLossOfClk
Loss of clock interrupt.

enumerator kRCM_IntLossOfLock
Loss of lock interrupt.

enumerator kRCM_IntWatchDog
Watch dog interrupt.

enumerator kRCM_IntExternalPin
External pin interrupt.

enumerator kRCM_IntGlobal
Global interrupts.

enumerator kRCM_IntCoreLockup
Core lock up interrupt

enumerator kRCM_IntSoftware
software interrupt

enumerator kRCM_IntStopModeAckErr
Stop mode ACK error interrupt.

enumerator kRCM_IntCore1
Core 1 interrupt.

enumerator kRCM_IntAll
Enable all interrupts.

typedef enum *_rcm_reset_source* rcm_reset_source_t
System Reset Source Name definitions.

typedef enum *_rcm_run_wait_filter_mode* rcm_run_wait_filter_mode_t
Reset pin filter select in Run and Wait modes.

typedef enum *_rcm_boot_rom_config* rcm_boot_rom_config_t
Boot from ROM configuration.

typedef enum *_rcm_reset_delay* rcm_reset_delay_t
Maximum delay time from interrupt asserts to system reset.

typedef enum *_rcm_interrupt_enable* rcm_interrupt_enable_t
System reset interrupt enable bit definitions.

typedef struct *_rcm_version_id* rcm_version_id_t
IP version ID definition.

typedef struct *_rcm_reset_pin_filter_config* rcm_reset_pin_filter_config_t

Reset pin filter configuration.

struct _rcm_version_id

#include <fsl_rcm.h> IP version ID definition.

Public Members

uint16_t feature

Feature Specification Number.

uint8_t minor

Minor version number.

uint8_t major

Major version number.

struct _rcm_reset_pin_filter_config

#include <fsl_rcm.h> Reset pin filter configuration.

Public Members

bool enableFilterInStop

Reset pin filter select in stop mode.

rcm_run_wait_filter_mode_t filterInRunWait

Reset pin filter in run/wait mode.

uint8_t busClockFilterCount

Reset pin bus clock filter width.

2.41 RNGA: Random Number Generator Accelerator Driver

FSL_RNGA_DRIVER_VERSION

RNGA driver version 2.0.2.

enum _rnga_mode

RNGA working mode.

Values:

enumerator kRNGA_ModeNormal

Normal Mode. The ring-oscillator clocks are active; RNGA generates entropy (randomness) from the clocks and stores it in shift registers.

enumerator kRNGA_ModeSleep

Sleep Mode. The ring-oscillator clocks are inactive; RNGA does not generate entropy.

typedef enum *_rnga_mode* rnga_mode_t

RNGA working mode.

void RNGA_Init(RNG_Type *base)

Initializes the RNGA.

This function initializes the RNGA. When called, the RNGA entropy generation starts immediately.

Parameters

- base – RNGA base address

void RNGA_Deinit(RNG_Type *base)

Shuts down the RNGA.

This function shuts down the RNGA.

Parameters

- base – RNGA base address

status_t RNGA_GetRandomData(RNG_Type *base, void *data, size_t data_size)

Gets random data.

This function gets random data from the RNGA.

Parameters

- base – RNGA base address
- data – pointer to user buffer to be filled by random data
- data_size – size of data in bytes

Returns

RNGA status

void RNGA_Seed(RNG_Type *base, uint32_t seed)

Feeds the RNGA module.

This function inputs an entropy value that the RNGA uses to seed its pseudo-random algorithm.

Parameters

- base – RNGA base address
- seed – input seed value

void RNGA_SetMode(RNG_Type *base, *rnga_mode_t* mode)

Sets the RNGA in normal mode or sleep mode.

This function sets the RNGA in sleep mode or normal mode.

Parameters

- base – RNGA base address
- mode – normal mode or sleep mode

rnga_mode_t RNGA_GetMode(RNG_Type *base)

Gets the RNGA working mode.

This function gets the RNGA working mode.

Parameters

- base – RNGA base address

Returns

normal mode or sleep mode

2.42 RTC: Real Time Clock

void RTC_Init(RTC_Type *base, const *rtc_config_t* *config)

Ungates the RTC clock and configures the peripheral for basic operation.

This function issues a software reset if the timer invalid flag is set.

Note: This API should be called at the beginning of the application using the RTC driver.

Parameters

- base – RTC peripheral base address
- config – Pointer to the user's RTC configuration structure.

static inline void RTC_Deinit(RTC_Type *base)

Stops the timer and gate the RTC clock.

Parameters

- base – RTC peripheral base address

void RTC_GetDefaultConfig(*rtc_config_t* *config)

Fills in the RTC config struct with the default settings.

The default values are as follows.

```
config->clockOutput = false;
config->wakeupSelect = false;
config->updateMode = false;
config->supervisorAccess = false;
config->compensationInterval = 0;
config->compensationTime = 0;
```

Parameters

- config – Pointer to the user's RTC configuration structure.

status_t RTC_SetDatetime(RTC_Type *base, const *rtc_datetime_t* *datetime)

Sets the RTC date and time according to the given time structure.

The RTC counter must be stopped prior to calling this function because writes to the RTC seconds register fail if the RTC counter is running.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to the structure where the date and time details 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)

Gets the RTC time and stores it in the given time structure.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to the structure where the date and time details are stored.

status_t RTC_SetAlarm(RTC_Type *base, const *rtc_datetime_t* *alarmTime)

Sets 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 the 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)

Returns the RTC alarm time.

Parameters

- base – RTC peripheral base address
- datetime – Pointer to the structure where the alarm date and time details are stored.

void RTC_EnableInterrupts(RTC_Type *base, uint32_t mask)

Enables the selected RTC interrupts.

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*

void RTC_DisableInterrupts(RTC_Type *base, uint32_t mask)

Disables the selected RTC interrupts.

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*

uint32_t RTC_GetEnabledInterrupts(RTC_Type *base)

Gets the enabled RTC interrupts.

Parameters

- base – RTC peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration *rtc_interrupt_enable_t*

uint32_t RTC_GetStatusFlags(RTC_Type *base)

Gets 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*

void RTC_ClearStatusFlags(RTC_Type *base, uint32_t mask)

Clears 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_EnableOscillatorClock(RTC_Type *base, bool enable)

Enable/Disable RTC 32kHz Oscillator clock.

Note: After setting this bit, wait the oscillator startup time before enabling the time counter to allow the 32.768 kHz clock time to stabilize.

Parameters

- base – RTC peripheral base address
- enable – Enable/Disable RTC 32.768 kHz clock

static inline void RTC_SetClockSource(RTC_Type *base)

Set RTC clock source.

Deprecated:

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

Note: After setting this bit, wait the oscillator startup time before enabling the time counter to allow the 32.768 kHz clock time to stabilize.

Parameters

- base – RTC peripheral base address

static inline void RTC_EnableLPOClock(RTC_Type *base, bool enable)

Enable/Disable RTC 1kHz LPO clock.

Note: After setting this bit, RTC prescaler increments using the LPO 1kHz clock and not the RTC 32kHz crystal clock.

Parameters

- base – RTC peripheral base address
- enable – Enable/Disable RTC 1kHz LPO clock

static inline void RTC_StartTimer(RTC_Type *base)

Starts the RTC time counter.

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.

RTC's seconds register can be written to only when the timer is stopped.

Parameters

- base – RTC peripheral base address

void RTC_GetMonotonicCounter(RTC_Type *base, uint64_t *counter)

Reads the values of the Monotonic Counter High and Monotonic Counter Low and returns them as a single value.

Parameters

- base – RTC peripheral base address
- counter – Pointer to variable where the value is stored.

void RTC_SetMonotonicCounter(RTC_Type *base, uint64_t counter)

Writes values Monotonic Counter High and Monotonic Counter Low by decomposing the given single value. The Monotonic Overflow Flag in RTC_SR is cleared due to the API.

Parameters

- base – RTC peripheral base address
- counter – Counter value

status_t RTC_IncrementMonotonicCounter(RTC_Type *base)

Increments the Monotonic Counter by one.

Increments the Monotonic Counter (registers RTC_MCLR and RTC_MCHR accordingly) by setting the monotonic counter enable (MER[MCE]) and then writing to the RTC_MCLR register. A write to the monotonic counter low that causes it to overflow also increments the monotonic counter high.

Parameters

- base – RTC peripheral base address

Returns

kStatus_Success: success kStatus_Fail: error occurred, either time invalid or monotonic overflow flag was found

FSL_RTC_DRIVER_VERSION

Version 2.4.0

enum _rtc_interrupt_enable

List of RTC interrupts.

Values:

enumerator kRTC_TimeInvalidInterruptEnable

Time invalid interrupt.

enumerator kRTC_TimeOverflowInterruptEnable
Time overflow interrupt.

enumerator kRTC_AlarmInterruptEnable
Alarm interrupt.

enumerator kRTC_MonotonicOverflowInterruptEnable
Monotonic Overflow Interrupt Enable

enumerator kRTC_SecondsInterruptEnable
Seconds interrupt.

enumerator kRTC_TestModeInterruptEnable

enumerator kRTC_FlashSecurityInterruptEnable

enumerator kRTC_TamperPinInterruptEnable

enumerator kRTC_SecurityModuleInterruptEnable

enumerator kRTC_LossOfClockInterruptEnable

enum _rtc_status_flags
List of RTC flags.
Values:

enumerator kRTC_TimeInvalidFlag
Time invalid flag

enumerator kRTC_TimeOverflowFlag
Time overflow flag

enumerator kRTC_AlarmFlag
Alarm flag

enumerator kRTC_MonotonicOverflowFlag
Monotonic Overflow Flag

enumerator kRTC_TamperInterruptDetectFlag
Tamper interrupt detect flag

enumerator kRTC_TestModeFlag

enumerator kRTC_FlashSecurityFlag

enumerator kRTC_TamperPinFlag

enumerator kRTC_SecurityTamperFlag

enumerator kRTC_LossOfClockTamperFlag

enum _rtc_osc_cap_load

List of RTC Oscillator capacitor load settings.

Values:

enumerator kRTC_Capacitor_2p

2 pF capacitor load

enumerator kRTC_Capacitor_4p

4 pF capacitor load

enumerator kRTC_Capacitor_8p

8 pF capacitor load

enumerator kRTC_Capacitor_16p

16 pF capacitor load

enum _rtc_timer_seconds_interrupt_frequency

List of RTC Timer Seconds Interrupt Frequencies.

Values:

enumerator kRTC_TimerSecondsFrequency1Hz

Timer seconds frequency is 1Hz

enumerator kRTC_TimerSecondsFrequency2Hz

Timer seconds frequency is 2Hz

enumerator kRTC_TimerSecondsFrequency4Hz

Timer seconds frequency is 4Hz

enumerator kRTC_TimerSecondsFrequency8Hz

Timer seconds frequency is 8Hz

enumerator kRTC_TimerSecondsFrequency16Hz

Timer seconds frequency is 16Hz

enumerator kRTC_TimerSecondsFrequency32Hz

Timer seconds frequency is 32Hz

enumerator kRTC_TimerSecondsFrequency64Hz

Timer seconds frequency is 64Hz

enumerator kRTC_TimerSecondsFrequency128Hz

Timer seconds frequency is 128Hz

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 enum *_rtc_osc_cap_load* rtc_osc_cap_load_t

List of RTC Oscillator capacitor load settings.

typedef enum *_rtc_timer_seconds_interrupt_frequency* rtc_timer_seconds_interrupt_frequency_t

List of RTC Timer Seconds Interrupt Frequencies.

typedef struct *_rtc_datetime* rtc_datetime_t

Structure is used to hold the date and time.

typedef struct *_rtc_pin_config* rtc_pin_config_t

RTC pin config structure.

typedef struct *_rtc_config* rtc_config_t

RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the *RTC_GetDefaultConfig()* function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

static inline uint32_t RTC_GetTamperTimeSeconds(RTC_Type *base)

Get the RTC tamper time seconds.

Parameters

- base – RTC peripheral base address

static inline void RTC_SetOscCapLoad(RTC_Type *base, uint32_t capLoad)

This function sets the specified capacitor configuration for the RTC oscillator.

Parameters

- base – RTC peripheral base address
- capLoad – Oscillator loads to enable. This is a logical OR of members of the enumeration *rtc_osc_cap_load_t*

static inline void RTC_Reset(RTC_Type *base)

Performs a software reset on the RTC module.

This resets all RTC registers except for the SWR bit and the RTC_WAR and RTC_RAR registers. The SWR bit is cleared by software explicitly clearing it.

Parameters

- base – RTC peripheral base address

static inline void RTC_EnableWakeUpPin(RTC_Type *base, bool enable)

Enables or disables the RTC Wakeup Pin Operation.

This function enable or disable RTC Wakeup Pin. The wakeup pin is optional and not available on all devices.

Parameters

- base – RTC_Type base pointer.
- enable – true to enable, false to disable.

static inline void RTC_EnableClockOutput(RTC_Type *base, bool enable)

Enables or disables the RTC 32 kHz clock output.

This function enables or disables the RTC 32 kHz clock output.

Parameters

- base – RTC_Type base pointer.
- enable – true to enable, false to disable.

void RTC_SetTimerSecondsInterruptFrequency(RTC_Type *base, *rtc_timer_seconds_interrupt_frequency_t* freq)

Sets the RTC timer seconds interrupt frequency.

This function sets the RTC timer seconds interrupt frequency.

Parameters

- base – RTC peripheral base address
- freq – The timer seconds interrupt frequency. This is a member of the enumeration *rtc_timer_seconds_interrupt_frequency_t*

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.

struct _rtc_pin_config

#include <fsl_rtc.h> RTC pin config structure.

Public Members

bool inputLogic

true: Tamper pin input data is logic one. false: Tamper pin input data is logic zero.

bool pinActiveLow

true: Tamper pin is active low. false: Tamper pin is active high.

bool filterEnable

true: Input filter is enabled on the tamper pin. false: Input filter is disabled on the tamper pin.

bool pullSelectNegate

true: Tamper pin pull resistor direction will negate the tamper pin. false: Tamper pin pull resistor direction will assert the tamper pin.

bool pullEnable

true: Pull resistor is enabled on tamper pin. false: Pull resistor is disabled on tamper pin.

struct __rtc__config

#include <fsl_rtc.h> RTC config structure.

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the *RTC_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 clockOutput

true: The 32 kHz clock is not output to other peripherals; false: The 32 kHz clock is output to other peripherals

bool wakeupSelect

true: Wakeup pin outputs the 32 KHz clock; false:Wakeup pin used to wakeup the chip

bool updateMode

true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked

bool supervisorAccess

true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported

uint32_t compensationInterval

Compensation interval that is written to the CIR field in RTC TCR Register

uint32_t compensationTime

Compensation time that is written to the TCR field in RTC TCR Register

2.43 SAI: Serial Audio Interface

2.44 SAI Driver

void SAI_Init(I2S_Type *base)

Initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_Init is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_Deinit(I2S_Type *base)

De-initializes the SAI peripheral.

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

- base – SAI base pointer.

void SAI_TxReset(I2S_Type *base)

Resets the SAI Tx.

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_RxReset(I2S_Type *base)

Resets the SAI Rx.

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

- base – SAI base pointer

void SAI_TxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Tx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Tx, false means disable.

void SAI_RxEnable(I2S_Type *base, bool enable)

Enables/disables the SAI Rx.

Parameters

- base – SAI base pointer.
- enable – True means enable SAI Rx, false means disable.

static inline void SAI_TxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetBitClockDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx bit clock direction.

Select bit clock direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_RxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Rx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

static inline void SAI_TxSetFrameSyncDirection(I2S_Type *base, *sai_master_slave_t* masterSlave)

Set Tx frame sync direction.

Select frame sync direction, master or slave.

Parameters

- base – SAI base pointer.
- masterSlave – reference sai_master_slave_t.

void SAI_TxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Transmitter bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_RxSetBitClockRate(I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)

Receiver bit clock rate configurations.

Parameters

- base – SAI base pointer.
- sourceClockHz – Bit clock source frequency.
- sampleRate – Audio data sample rate.
- bitWidth – Audio data bitWidth.
- channelNumbers – Audio channel numbers.

void SAI_TxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Transmitter Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_RxSetBitclockConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_bit_clock_t* *config)

Receiver Bit clock configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – bit clock other configurations, can be NULL in slave mode.

void SAI_SetMasterClockConfig(I2S_Type *base, *sai_master_clock_t* *config)

Master clock configurations.

Parameters

- base – SAI base pointer.
- config – master clock configurations.

void SAI_TxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI transmitter fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_RxSetFifoConfig(I2S_Type *base, *sai_fifo_t* *config)

SAI receiver fifo configurations.

Parameters

- base – SAI base pointer.
- config – fifo configurations.

void SAI_TxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI transmitter Frame sync configurations.

Parameters

- base – SAI base pointer.
- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_RxSetFrameSyncConfig(I2S_Type *base, *sai_master_slave_t* masterSlave, *sai_frame_sync_t* *config)

SAI receiver Frame sync configurations.

Parameters

- base – SAI base pointer.

- masterSlave – master or slave.
- config – frame sync configurations, can be NULL in slave mode.

void SAI_TxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI transmitter Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_RxSetSerialDataConfig(I2S_Type *base, *sai_serial_data_t* *config)

SAI receiver Serial data configurations.

Parameters

- base – SAI base pointer.
- config – serial data configurations.

void SAI_TxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI transmitter configurations.

Parameters

- base – SAI base pointer.
- config – transmitter configurations.

void SAI_RxSetConfig(I2S_Type *base, *sai_transceiver_t* *config)

SAI receiver configurations.

Parameters

- base – SAI base pointer.
- config – receiver configurations.

void SAI_GetClassicI2SConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get classic I2S mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetLeftJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get left justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetRightJustifiedConfig(*sai_transceiver_t* *config, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get right justified mode configurations.

Parameters

- config – transceiver configurations.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetTDMConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, uint32_t dataWordNum, uint32_t saiChannelMask)

Get TDM mode configurations.

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data word width.
- dataWordNum – word number in one frame.
- saiChannelMask – mask value of the channel to be enable.

void SAI_GetDSPConfig(*sai_transceiver_t* *config, *sai_frame_sync_len_t* frameSyncWidth, *sai_word_width_t* bitWidth, *sai_mono_stereo_t* mode, uint32_t saiChannelMask)

Get DSP mode configurations.

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
SAI_TxSetConfig(base, config)
```

Note: DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI_RxSetConfig instead of SAI_TxSetConfig:

```
SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth, kSAI_Stereo, channelMask)
config->frameSync.frameSyncEarly    = true;
SAI_TxSetConfig(base, config)
```

Parameters

- config – transceiver configurations.
- frameSyncWidth – length of frame sync.
- bitWidth – audio data bitWidth.
- mode – audio data channel.
- saiChannelMask – mask value of the channel to enable.

static inline uint32_t SAI_TxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Tx status flag value. Use the Status Mask to get the status value needed.

static inline void SAI_TxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Tx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following source if defined:
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

static inline uint32_t SAI_RxGetStatusFlag(I2S_Type *base)

Gets the SAI Tx status flag state.

Parameters

- base – SAI base pointer

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

static inline void SAI_RxClearStatusFlags(I2S_Type *base, uint32_t mask)

Clears the SAI Rx status flag state.

Parameters

- base – SAI base pointer
- mask – State mask. It can be a combination of the following sources if defined.
  - kSAI_WordStartFlag
  - kSAI_SyncErrorFlag
  - kSAI_FIFOErrorFlag

void SAI_TxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_RxSoftwareReset(I2S_Type *base, *sai_reset_type_t* resetType)

Do software reset or FIFO reset .

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

- base – SAI base pointer
- resetType – Reset type, FIFO reset or software reset

void SAI_TxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Tx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_RxSetChannelFIFOMask(I2S_Type *base, uint8_t mask)

Set the Rx channel FIFO enable mask.

Parameters

- base – SAI base pointer
- mask – Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

void SAI_TxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Tx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_RxSetDataOrder(I2S_Type *base, *sai_data_order_t* order)

Set the Rx data order.

Parameters

- base – SAI base pointer
- order – Data order MSB or LSB

void SAI_TxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetBitClockPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Tx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_RxSetFrameSyncPolarity(I2S_Type *base, *sai_clock_polarity_t* polarity)

Set the Rx data order.

Parameters

- base – SAI base pointer
- polarity –

void SAI_TxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Tx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

void SAI_RxSetFIFOPacking(I2S_Type *base, *sai_fifo_packing_t* pack)

Set Rx FIFO packing feature.

Parameters

- base – SAI base pointer.
- pack – FIFO pack type. It is element of sai_fifo_packing_t.

static inline void SAI_TxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Tx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_RxSetFIFOErrorContinue(I2S_Type *base, bool isEnabled)

Set Rx FIFO error continue.

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

- base – SAI base pointer.
- isEnabled – Is FIFO error continue enabled, true means enable, false means disable.

static inline void SAI_TxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.

- kSAI_WordStartInterruptEnable
- kSAI_SyncErrorInterruptEnable
- kSAI_FIFOWarningInterruptEnable
- kSAI_FIFORequestInterruptEnable
- kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxEnableInterrupts(I2S_Type *base, uint32_t mask)

Enables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Tx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_RxDisableInterrupts(I2S_Type *base, uint32_t mask)

Disables the SAI Rx interrupt requests.

Parameters

- base – SAI base pointer
- mask – interrupt source The parameter can be a combination of the following sources if defined.
  - kSAI_WordStartInterruptEnable
  - kSAI_SyncErrorInterruptEnable
  - kSAI_FIFOWarningInterruptEnable
  - kSAI_FIFORequestInterruptEnable
  - kSAI_FIFOErrorInterruptEnable

static inline void SAI_TxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Tx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline void SAI_RxEnableDMA(I2S_Type *base, uint32_t mask, bool enable)

Enables/disables the SAI Rx DMA requests.

Parameters

- base – SAI base pointer
- mask – DMA source The parameter can be a combination of the following sources if defined.
  - kSAI_FIFOWarningDMAEnable
  - kSAI_FIFORequestDMAEnable
- enable – True means enable DMA, false means disable DMA.

static inline uintptr_t SAI_TxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Tx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

static inline uintptr_t SAI_RxGetDataRegisterAddress(I2S_Type *base, uint32_t channel)

Gets the SAI Rx data register address.

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

- base – SAI base pointer.
- channel – Which data channel used.

Returns

data register address.

void SAI_WriteBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.

- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

void SAI_WriteMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Sends data to multi channel using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be written.
- size – Bytes to be written.

static inline void SAI_WriteData(I2S_Type *base, uint32_t channel, uint32_t data)

Writes data into SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- data – Data needs to be written.

void SAI_ReadBlocking(I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

void SAI_ReadMultiChannelBlocking(I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

Receives multi channel data using a blocking method.

Note: This function blocks by polling until data is ready to be sent.

Parameters

- base – SAI base pointer.
- channel – Data channel used.
- channelMask – channel mask.
- bitWidth – How many bits in an audio word; usually 8/16/24/32 bits.
- buffer – Pointer to the data to be read.
- size – Bytes to be read.

static inline uint32_t SAI_ReadData(I2S_Type *base, uint32_t channel)

Reads data from the SAI FIFO.

Parameters

- base – SAI base pointer.
- channel – Data channel used.

Returns

Data in SAI FIFO.

void SAI_TransferTxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Tx handle.

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function

void SAI_TransferRxCreateHandle(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_callback_t* callback, void *userData)

Initializes the SAI Rx handle.

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- callback – Pointer to the user callback function.
- userData – User parameter passed to the callback function.

void SAI_TransferTxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI transmitter transfer configurations.

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.

- config – tranmitter configurations.

void SAI_TransferRxSetConfig(I2S_Type *base, *sai_handle_t* *handle, *sai_transceiver_t* *config)

SAI receiver transfer configurations.

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

- base – SAI base pointer.
- handle – SAI handle pointer.
- config – receiver configurations.

status_t SAI_TransferSendNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking send transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_TxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferReceiveNonBlocking(I2S_Type *base, *sai_handle_t* *handle, *sai_transfer_t* *xfer)

Performs an interrupt non-blocking receive transfer on SAI.

Note: This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- xfer – Pointer to the sai_transfer_t structure.

Return values

- kStatus_Success – Successfully started the data receive.
- kStatus_SAI_RxBusy – Previous receive still not finished.
- kStatus_InvalidArgument – The input parameter is invalid.

status_t SAI_TransferGetSendCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a set byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count sent.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

status_t SAI_TransferGetReceiveCount(I2S_Type *base, *sai_handle_t* *handle, size_t *count)

Gets a received byte count.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.
- count – Bytes count received.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is not a non-blocking transaction currently in progress.

void SAI_TransferAbortSend(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current send.

Note: This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferAbortReceive(I2S_Type *base, *sai_handle_t* *handle)

Aborts the current IRQ receive.

Note: This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

- base – SAI base pointer
- handle – Pointer to the sai_handle_t structure which stores the transfer state.

void SAI_TransferTerminateSend(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceive(I2S_Type *base, *sai_handle_t* *handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_TransferRxHandleIRQ(I2S_Type *base, *sai_handle_t* *handle)

Tx interrupt handler.

Parameters

- base – SAI base pointer.
- handle – Pointer to the sai_handle_t structure.

void SAI_DriverIRQHandler(uint32_t instance)

SAI driver IRQ handler common entry.

This function provides the common IRQ request entry for SAI.

Parameters

- instance – SAI instance.

FSL_SAI_DRIVER_VERSION

Version 2.4.11

_sai_status_t, SAI return status.

Values:

enumerator kStatus_SAI_TxBusy

SAI Tx is busy.

enumerator kStatus_SAI_RxBusy

SAI Rx is busy.

enumerator kStatus_SAI_TxError
SAI Tx FIFO error.

enumerator kStatus_SAI_RxError
SAI Rx FIFO error.

enumerator kStatus_SAI_QueueFull
SAI transfer queue is full.

enumerator kStatus_SAI_TxIdle
SAI Tx is idle

enumerator kStatus_SAI_RxIdle
SAI Rx is idle

_sai_channel_mask,.sai channel mask value, actual channel numbers is depend soc specific
Values:

enumerator kSAI_Channel0Mask
channel 0 mask value

enumerator kSAI_Channel1Mask
channel 1 mask value

enumerator kSAI_Channel2Mask
channel 2 mask value

enumerator kSAI_Channel3Mask
channel 3 mask value

enumerator kSAI_Channel4Mask
channel 4 mask value

enumerator kSAI_Channel5Mask
channel 5 mask value

enumerator kSAI_Channel6Mask
channel 6 mask value

enumerator kSAI_Channel7Mask
channel 7 mask value

enum _sai_protocol
Define the SAI bus type.
Values:

enumerator kSAI_BusLeftJustified
Uses left justified format.

enumerator kSAI_BusRightJustified
Uses right justified format.

enumerator kSAI_BusI2S
Uses I2S format.

enumerator kSAI_BusPCMA
Uses I2S PCM A format.

enumerator kSAI_BusPCMB
Uses I2S PCM B format.

enum _sai_master_slave
Master or slave mode.
Values:

enumerator kSAI_Master
Master mode include bclk and frame sync

enumerator kSAI_Slave
Slave mode include bclk and frame sync

enumerator kSAI_Bclk_Master_FrameSync_Slave
bclk in master mode, frame sync in slave mode

enumerator kSAI_Bclk_Slave_FrameSync_Master
bclk in slave mode, frame sync in master mode

enum _sai_mono_stereo
Mono or stereo audio format.
Values:

enumerator kSAI_Stereo
Stereo sound.

enumerator kSAI_MonoRight
Only Right channel have sound.

enumerator kSAI_MonoLeft
Only left channel have sound.

enum _sai_data_order
SAI data order, MSB or LSB.
Values:

enumerator kSAI_DataLSB
LSB bit transferred first

enumerator kSAI_DataMSB
MSB bit transferred first

enum _sai_clock_polarity
SAI clock polarity, active high or low.
Values:

enumerator kSAI_PolarityActiveHigh
Drive outputs on rising edge

enumerator kSAI_PolarityActiveLow
Drive outputs on falling edge

enumerator kSAI_SampleOnFallingEdge
Sample inputs on falling edge

enumerator kSAI_SampleOnRisingEdge
Sample inputs on rising edge

enum _sai_sync_mode
Synchronous or asynchronous mode.
Values:

enumerator kSAI_ModeAsync
Asynchronous mode

enumerator kSAI_ModeSync
Synchronous mode (with receiver or transmit)

enum _sai_mclk_source
Mater clock source.
Values:

enumerator kSAI_MclkSourceSysclk
Master clock from the system clock

enumerator kSAI_MclkSourceSelect1
Master clock from source 1

enumerator kSAI_MclkSourceSelect2
Master clock from source 2

enumerator kSAI_MclkSourceSelect3
Master clock from source 3

enum _sai_bclk_source

Bit clock source.

Values:

enumerator kSAI_BclkSourceBusclk

Bit clock using bus clock

enumerator kSAI_BclkSourceMclkOption1

Bit clock MCLK option 1

enumerator kSAI_BclkSourceMclkOption2

Bit clock MCLK option2

enumerator kSAI_BclkSourceMclkOption3

Bit clock MCLK option3

enumerator kSAI_BclkSourceMclkDiv

Bit clock using master clock divider

enumerator kSAI_BclkSourceOtherSai0

Bit clock from other SAI device

enumerator kSAI_BclkSourceOtherSai1

Bit clock from other SAI device

_sai_interrupt_enable_t, The SAI interrupt enable flag

Values:

enumerator kSAI_WordStartInterruptEnable

Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorInterruptEnable

Sync error flag, means the sync error is detected

enumerator kSAI_FIFOWarningInterruptEnable

FIFO warning flag, means the FIFO is empty

enumerator kSAI_FIFOErrorInterruptEnable

FIFO error flag

enumerator kSAI_FIFORequestInterruptEnable

FIFO request, means reached watermark

_sai_dma_enable_t, The DMA request sources

Values:

enumerator kSAI_FIFOWarningDMAEnable
FIFO warning caused by the DMA request

enumerator kSAI_FIFORequestDMAEnable
FIFO request caused by the DMA request

_sai_flags, The SAI status flag

Values:

enumerator kSAI_WordStartFlag
Word start flag, means the first word in a frame detected

enumerator kSAI_SyncErrorFlag
Sync error flag, means the sync error is detected

enumerator kSAI_FIFOErrorFlag
FIFO error flag

enumerator kSAI_FIFORequestFlag
FIFO request flag.

enumerator kSAI_FIFOWarningFlag
FIFO warning flag

enum _sai_reset_type
The reset type.

Values:

enumerator kSAI_ResetTypeSoftware
Software reset, reset the logic state

enumerator kSAI_ResetTypeFIFO
FIFO reset, reset the FIFO read and write pointer

enumerator kSAI_ResetAll
All reset.

enum _sai_fifo_packing
The SAI packing mode The mode includes 8 bit and 16 bit packing.

Values:

enumerator kSAI_FifoPackingDisabled
Packing disabled

enumerator kSAI_FifoPacking8bit
8 bit packing enabled

enumerator kSAI_FifoPacking16bit
16bit packing enabled

enum _sai_sample_rate
Audio sample rate.
Values:

enumerator kSAI_SampleRate8KHz
Sample rate 8000 Hz

enumerator kSAI_SampleRate11025Hz
Sample rate 11025 Hz

enumerator kSAI_SampleRate12KHz
Sample rate 12000 Hz

enumerator kSAI_SampleRate16KHz
Sample rate 16000 Hz

enumerator kSAI_SampleRate22050Hz
Sample rate 22050 Hz

enumerator kSAI_SampleRate24KHz
Sample rate 24000 Hz

enumerator kSAI_SampleRate32KHz
Sample rate 32000 Hz

enumerator kSAI_SampleRate44100Hz
Sample rate 44100 Hz

enumerator kSAI_SampleRate48KHz
Sample rate 48000 Hz

enumerator kSAI_SampleRate96KHz
Sample rate 96000 Hz

enumerator kSAI_SampleRate192KHz
Sample rate 192000 Hz

enumerator kSAI_SampleRate384KHz
Sample rate 384000 Hz

enum _sai_word_width
Audio word width.
Values:

enumerator kSAI_WordWidth8bits
Audio data width 8 bits

enumerator kSAI_WordWidth16bits
Audio data width 16 bits

enumerator kSAI_WordWidth24bits
Audio data width 24 bits

enumerator kSAI_WordWidth32bits
Audio data width 32 bits

enum _sai_data_pin_state
sai data pin state definition
Values:

enumerator kSAI_DataPinStateTriState
transmit data pins are tri-stated when slots are masked or channels are disabled

enumerator kSAI_DataPinStateOutputZero
transmit data pins are never tri-stated and will output zero when slots are masked or channel disabled

enum _sai_fifo_combine
sai fifo combine mode definition
Values:

enumerator kSAI_FifoCombineDisabled
sai TX/RX fifo combine mode disabled

enumerator kSAI_FifoCombineModeEnabledOnRead
sai TX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnWrite
sai TX fifo combine mode enabled on FIFO write

enumerator kSAI_RxFifoCombineModeEnabledOnWrite
sai RX fifo combine mode enabled on FIFO write

enumerator kSAI_RXFifoCombineModeEnabledOnRead
sai RX fifo combine mode enabled on FIFO reads

enumerator kSAI_FifoCombineModeEnabledOnReadWrite
sai TX/RX fifo combined mode enabled on FIFO read/writes

enum _sai_transceiver_type
sai transceiver type
Values:

enumerator kSAI_Transmitter
sai transmitter

enumerator kSAI_Receiver
sai receiver

enum _sai_frame_sync_len
sai frame sync len
Values:

enumerator kSAI_FrameSyncLenOneBitClk
1 bit clock frame sync len for DSP mode

enumerator kSAI_FrameSyncLenPerWordWidth
Frame sync length decided by word width

typedef enum *_sai_protocol* sai_protocol_t
Define the SAI bus type.

typedef enum *_sai_master_slave* sai_master_slave_t
Master or slave mode.

typedef enum *_sai_mono_stereo* sai_mono_stereo_t
Mono or stereo audio format.

typedef enum *_sai_data_order* sai_data_order_t
SAI data order, MSB or LSB.

typedef enum *_sai_clock_polarity* sai_clock_polarity_t
SAI clock polarity, active high or low.

typedef enum *_sai_sync_mode* sai_sync_mode_t
Synchronous or asynchronous mode.

typedef enum *_sai_mclk_source* sai_mclk_source_t
Mater clock source.

typedef enum *_sai_bclk_source* sai_bclk_source_t
Bit clock source.

typedef enum *_sai_reset_type* sai_reset_type_t
The reset type.

typedef enum *_sai_fifo_packing* sai_fifo_packing_t
The SAI packing mode The mode includes 8 bit and 16 bit packing.

typedef struct *_sai_config* sai_config_t
SAI user configuration structure.

typedef enum *_sai_sample_rate* sai_sample_rate_t
Audio sample rate.

typedef enum *_sai_word_width* sai_word_width_t
Audio word width.

typedef enum *_sai_data_pin_state* sai_data_pin_state_t
sai data pin state definition

typedef enum *_sai_fifo_combine* sai_fifo_combine_t
sai fifo combine mode definition

typedef enum *_sai_transceiver_type* sai_transceiver_type_t
sai transceiver type

typedef enum *_sai_frame_sync_len* sai_frame_sync_len_t
sai frame sync len

typedef struct *_sai_transfer_format* sai_transfer_format_t
sai transfer format

typedef struct *_sai_master_clock* sai_master_clock_t
master clock configurations

typedef struct *_sai_fifo* sai_fifo_t
sai fifo configurations

typedef struct *_sai_bit_clock* sai_bit_clock_t
sai bit clock configurations

typedef struct *_sai_frame_sync* sai_frame_sync_t
sai frame sync configurations

typedef struct *_sai_serial_data* sai_serial_data_t
sai serial data configurations

typedef struct *_sai_transceiver* sai_transceiver_t
sai transceiver configurations

typedef struct *_sai_transfer* sai_transfer_t
SAI transfer structure.

typedef struct *_sai_handle* sai_handle_t

typedef void (*sai_transfer_callback_t)(I2S_Type *base, *sai_handle_t* *handle, *status_t* status, void *userData)
SAI transfer callback prototype.

MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK

Used to control whether *SAI_RxSetFifoConfig()*/SAI_TxSetFifoConfig() allows a NULL FIFO watermark.

If this macro is set to 0 then *SAI_RxSetFifoConfig()*/SAI_TxSetFifoConfig() will set the watermark to half of the FIFO's depth if passed a NULL watermark.

MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG

Disable implicit channel data configuration within *SAI_TxSetConfig()*/SAI_RxSetConfig().

Use this macro to control whether *SAI_RxSetConfig()*/SAI_TxSetConfig() will attempt to implicitly configure the channel data. By channel data we mean the startChannel, channelMask, endChannel, and channelNums fields from the sai_transciever_t structure. By default, *SAI_TxSetConfig()*/SAI_RxSetConfig() will attempt to compute these fields, which may not be desired in cases where the user wants to set them before the call to said functions.

SAI_XFER_QUEUE_SIZE

SAI transfer queue size, user can refine it according to use case.

FSL_SAI_HAS_FIFO_EXTEND_FEATURE

sai fifo feature

struct _sai_config

#include <fsl_sai.h> SAI user configuration structure.

Public Members

sai_protocol_t protocol

Audio bus protocol in SAI

sai_sync_mode_t syncMode

SAI sync mode, control Tx/Rx clock sync

bool mclkOutputEnable

Master clock output enable, true means master clock divider enabled

sai_mclk_source_t mclkSource

Master Clock source

sai_bclk_source_t bclkSource

Bit Clock source

sai_master_slave_t masterSlave

Master or slave

struct _sai_transfer_format

#include <fsl_sai.h> sai transfer format

Public Members

uint32_t sampleRate_Hz
: Sample rate of audio data

uint32_t bitWidth
: Data length of audio data, usually 8/16/24/32 bits

sai_mono_stereo_t stereo
: Mono or stereo

uint32_t masterClockHz
: Master clock frequency in Hz

uint8_t watermark
: Watermark value

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_protocol_t protocol
: Which audio protocol used

bool isFrameSyncCompact
: True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

struct __sai_master_clock
: *#include <fsl_sai.h>* master clock configurations

Public Members

bool mclkOutputEnable
: master clock output enable

sai_mclk_source_t mclkSource
: Master Clock source

uint32_t mclkHz
target mclk frequency

uint32_t mclkSourceClkHz
mclk source frequency

struct _sai_fifo
#include <fsl_sai.h> sai fifo configurations

Public Members

bool fifoContinueOneError
fifo continues when error occur

sai_fifo_combine_t fifoCombine
fifo combine mode

sai_fifo_packing_t fifoPacking
fifo packing mode

uint8_t fifoWatermark
fifo watermark

struct _sai_bit_clock
#include <fsl_sai.h> sai bit clock configurations

Public Members

bool bclkInputDelay
bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .

sai_clock_polarity_t bclkPolarity
bit clock polarity

sai_bclk_source_t bclkSource
bit Clock source

struct _sai_frame_sync
#include <fsl_sai.h> sai frame sync configurations

Public Members

uint8_t frameSyncWidth
frame sync width in number of bit clocks

bool frameSyncEarly

TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame

bool frameSyncGenerateOnDemand

internal frame sync is generated when FIFO waring flag is clear

sai_clock_polarity_t frameSyncPolarity

frame sync polarity

struct __sai__serial__data

#include <fsl_sai.h> sai serial data configurations

Public Members

sai_data_pin_state_t dataMode

sai data pin state when slots masked or channel disabled

sai_data_order_t dataOrder

configure whether the LSB or MSB is transmitted first

uint8_t dataWord0Length

configure the number of bits in the first word in each frame

uint8_t dataWordNLength

configure the number of bits in the each word in each frame, except the first word

uint8_t dataWordLength

used to record the data length for dma transfer

uint8_t dataFirstBitShifted

Configure the bit index for the first bit transmitted for each word in the frame

uint8_t dataWordNum

configure the number of words in each frame

uint32_t dataMaskedWord

configure whether the transmit word is masked

struct __sai__transceiver

#include <fsl_sai.h> sai transceiver configurations

Public Members

sai_serial_data_t serialData

serial data configurations

sai_frame_sync_t frameSync
ws configurations

sai_bit_clock_t bitClock
bit clock configurations

sai_fifo_t fifo
fifo configurations

sai_master_slave_t masterSlave
transceiver is master or slave

sai_sync_mode_t syncMode
transceiver sync mode

uint8_t startChannel
Transfer start channel

uint8_t channelMask
enabled channel mask value, reference _sai_channel_mask

uint8_t endChannel
end channel number

uint8_t channelNums
Total enabled channel numbers

struct __sai__transfer
#include <fsl_sai.h> SAI transfer structure.

Public Members

uint8_t *data
Data start address to transfer.

size_t dataSize
Transfer size.

struct __sai__handle
#include <fsl_sai.h> SAI handle structure.

Public Members

I2S_Type *base
base address

uint32_t state
: Transfer status

sai_transfer_callback_t callback
: Callback function called at transfer event

void *userData
: Callback parameter passed to callback function

uint8_t bitWidth
: Bit width for transfer, 8/16/24/32 bits

uint8_t channel
: Transfer start channel

uint8_t channelMask
: enabled channel mask value, refernece _sai_channel_mask

uint8_t endChannel
: end channel number

uint8_t channelNums
: Total enabled channel numbers

sai_transfer_t saiQueue[(4U)]
: Transfer queue storing queued transfer

size_t transferSize[(4U)]
: Data bytes need to transfer

volatile uint8_t queueUser
: Index for user to queue transfer

volatile uint8_t queueDriver
: Index for driver to get the transfer data and size

uint8_t watermark
: Watermark value

2.45 SAI EDMA Driver

void SAI_TransferTxCreateHandleEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_edma_callback_t* callback, void *userData, *edma_handle_t* *txDmaHandle)

Initializes the SAI eDMA handle.

This function initializes the SAI master DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- txDmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void SAI_TransferRxCreateHandleEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_edma_callback_t* callback, void *userData, *edma_handle_t* *rxDmaHandle)

Initializes the SAI Rx eDMA handle.

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- callback – Pointer to user callback function.
- userData – User parameter passed to the callback function.
- rxDmaHandle – eDMA handle pointer, this handle shall be static allocated by users.

void SAI_TransferSetInterleaveType(*sai_edma_handle_t* *handle, *sai_edma_interleave_t* interleaveType)

Initializes the SAI interleave type.

This function initializes the SAI DMA handle member interleaveType, it shall be called only when application would like to use type kSAI_EDMAInterleavePerChannelBlock, since the default interleaveType is kSAI_EDMAInterleavePerChannelSample always

Parameters

- handle – SAI eDMA handle pointer.
- interleaveType – Multi channel interleave type.

void SAI_TransferTxSetConfigEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transceiver_t* *saiConfig)

Configures the SAI Tx.

Note: SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of sai_transceiver_t with the corresponding values of _sai_channel_mask enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnWrite to the fifoCombine member of sai_fifo_combine_t which is a member of sai_transceiver_t. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_
↪Channel1Mask);
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnWrite;
SAI_TransferTxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

void SAI_TransferRxSetConfigEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transceiver_t* *saiConfig)

Configures the SAI Rx.

Note: SAI eDMA supports data transfer in a multiple SAI channels if the FIFO Combine feature is supported. To activate the multi-channel transfer enable SAI channels by filling the channelMask of sai_transceiver_t with the corresponding values of _sai_channel_mask enum, enable the FIFO Combine mode by assigning kSAI_FifoCombineModeEnabledOnRead to the fifoCombine member of sai_fifo_combine_t which is a member of sai_transceiver_t. This is an example of multi-channel data transfer configuration step.

```
sai_transceiver_t config;
SAI_GetClassicI2SConfig(&config, kSAI_WordWidth16bits, kSAI_Stereo, kSAI_Channel0Mask|kSAI_
↪Channel1Mask);
config.fifo.fifoCombine = kSAI_FifoCombineModeEnabledOnRead;
SAI_TransferRxSetConfigEDMA(I2S0, &edmaHandle, &config);
```

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- saiConfig – sai configurations.

status_t SAI_TransferSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI transfer using DMA.

This function support multi channel transfer,

a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers

b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Note: This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure.

Return values

- kStatus_Success – Start a SAI eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_TxBusy – SAI is busy sending data.

status_t SAI_TransferReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer)

Performs a non-blocking SAI receive using eDMA.

This function support multi channel transfer,

a. for the sai IP support fifo combine mode, application should enable the fifo combine mode, no limitation on channel numbers
b. for the sai IP not support fifo combine mode, sai edma provide another solution which using EDMA modulo feature, but support 2 or 4 channels only.

Note: This interface returns immediately after the transfer initiates. Call the SAI_GetReceiveRemainingBytes to poll the transfer status and check whether the SAI transfer is finished.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.
- xfer – Pointer to DMA transfer structure.

Return values

- kStatus_Success – Start a SAI eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.
- kStatus_RxBusy – SAI is busy receiving data.

status_t SAI_TransferSendLoopEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer, uint32_t loopTransferCount)

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function SAI_TransferAbortSendEDMA to stop the loop transfer.

Note: This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

- xfer – Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).
- loopTransferCount – the counts of xfer array.

Return values

- kStatus_Success – Start a SAI eDMA send successfully.
- kStatus_InvalidArgument – The input argument is invalid.

status_t SAI_TransferReceiveLoopEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, *sai_transfer_t* *xfer, uint32_t loopTransferCount)

Performs a non-blocking SAI loop transfer using eDMA.

Once the loop transfer start, application can use function SAI_TransferAbortReceiveEDMA to stop the loop transfer.

Note: This function support loop transfer only,such as A->B->...->A, application must be aware of that the more counts of the loop transfer, then more tcd memory required, as the function use the tcd pool in sai_edma_handle_t, so application could redefine the SAI_XFER_QUEUE_SIZE to determine the proper TCD pool size. This function support one sai channel only.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- xfer – Pointer to the DMA transfer structure, should be a array with elements counts >=1(loopTransferCount).
- loopTransferCount – the counts of xfer array.

Return values

- kStatus_Success – Start a SAI eDMA receive successfully.
- kStatus_InvalidArgument – The input argument is invalid.

void SAI_TransferTerminateSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Terminate all SAI send.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSendEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferTerminateReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Terminate all SAI receive.

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceiveEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferAbortSendEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Aborts a SAI transfer using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateSendEDMA.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.

void SAI_TransferAbortReceiveEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Aborts a SAI receive using eDMA.

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateReceiveEDMA.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.

status_t SAI_TransferGetSendCountEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, size_t *count)

Gets byte count sent by SAI.

Parameters

- base – SAI base pointer.
- handle – SAI eDMA handle pointer.
- count – Bytes count sent by SAI.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

status_t SAI_TransferGetReceiveCountEDMA(I2S_Type *base, *sai_edma_handle_t* *handle, size_t *count)

Gets byte count received by SAI.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.
- count – Bytes count received by SAI.

Return values

- kStatus_Success – Succeed get the transfer count.
- kStatus_NoTransferInProgress – There is no non-blocking transaction in progress.

uint32_t SAI_TransferGetValidTransferSlotsEDMA(I2S_Type *base, *sai_edma_handle_t* *handle)

Gets valid transfer slot.

This function can be used to query the valid transfer request slot that the application can submit. It should be called in the critical section, that means the application could call it

in the corresponding callback function or disable IRQ before calling it in the application, otherwise, the returned value may not correct.

Parameters

- base – SAI base pointer
- handle – SAI eDMA handle pointer.

Return values

valid – slot count that application submit.

FSL_SAI_EDMA_DRIVER_VERSION

Version 2.7.4

enum _sai_edma_interleave

sai interleave type

Values:

enumerator kSAI_EDMAInterleavePerChannelSample

enumerator kSAI_EDMAInterleavePerChannelBlock

typedef struct *sai_edma_handle* sai_edma_handle_t

typedef void (*sai_edma_callback_t)(I2S_Type *base, *sai_edma_handle_t* *handle, *status_t* status, void *userData)

SAI eDMA transfer callback function for finish and error.

typedef enum *_sai_edma_interleave* sai_edma_interleave_t

sai interleave type

MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL

the SAI enable position When calling SAI_TransferReceiveEDMA

MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL

the SAI enable position When calling SAI_TransferSendEDMA

struct sai_edma_handle

#include <fsl_sai_edma.h> SAI DMA transfer handle, users should not touch the content of the handle.

Public Members

edma_handle_t *dmaHandle

DMA handler for SAI send

uint8_t nbytes

eDMA minor byte transfer count initially configured.

uint8_t bytesPerFrame
: Bytes in a frame

uint8_t channelMask
: Enabled channel mask value, reference _sai_channel_mask

uint8_t channelNums
: total enabled channel nums

uint8_t channel
: Which data channel

uint8_t count
: The transfer data count in a DMA request

uint32_t state
: Internal state for SAI eDMA transfer

sai_edma_callback_t callback
: Callback for users while transfer finish or error occurs

void *userData
: User callback parameter

uint8_t tcd[((4U) + 1U) * sizeof(*edma_tcd_t*)]
: TCD pool for eDMA transfer.

sai_transfer_t saiQueue[(4U)]
: Transfer queue storing queued transfer.

size_t transferSize[(4U)]
: Data bytes need to transfer

sai_edma_interleave_t interleaveType
: Transfer interleave type

volatile uint8_t queueUser
: Index for user to queue transfer.

volatile uint8_t queueDriver
: Index for driver to get the transfer data and size

2.46 SIM: System Integration Module Driver

FSL_SIM_DRIVER_VERSION
: Driver version.

enum _sim_usb_volt_reg_enable_mode

USB voltage regulator enable setting.

Values:

enumerator kSIM_UsbVoltRegEnable

Enable voltage regulator.

enumerator kSIM_UsbVoltRegEnableInLowPower

Enable voltage regulator in VLPR/VLPW modes.

enumerator kSIM_UsbVoltRegEnableInStop

Enable voltage regulator in STOP/VLPS/LLS/VLLS modes.

enumerator kSIM_UsbVoltRegEnableInAllModes

Enable voltage regulator in all power modes.

enum _sim_flash_mode

Flash enable mode.

Values:

enumerator kSIM_FlashDisableInWait

Disable flash in wait mode.

enumerator kSIM_FlashDisable

Disable flash in normal mode.

typedef struct *_sim_uid* sim_uid_t

Unique ID.

void SIM_SetUsbVoltRegulatorEnableMode(uint32_t mask)

Sets the USB voltage regulator setting.

This function configures whether the USB voltage regulator is enabled in normal RUN mode, STOP/VLPS/LLS/VLLS modes, and VLPR/VLPW modes. The configurations are passed in as mask value of *_sim_usb_volt_reg_enable_mode*. For example, to enable USB voltage regulator in RUN/VLPR/VLPW modes and disable in STOP/VLPS/LLS/VLLS mode, use:

SIM_SetUsbVoltRegulatorEnableMode(kSIM_UsbVoltRegEnable | kSIM_UsbVoltRegEnableInLowPower);

Parameters

- mask – USB voltage regulator enable setting.

void SIM_GetUniqueId(*sim_uid_t* *uid)

Gets the unique identification register value.

Parameters

- uid – Pointer to the structure to save the UID value.

static inline void SIM_SetFlashMode(uint8_t mode)

Sets the flash enable mode.

Parameters

- mode – The mode to set; see *_sim_flash_mode* for mode details.

struct __sim__uid
#include <fsl_sim.h> Unique ID.

Public Members

uint32_t H
UIDH.

uint32_t M
SIM_UIDM.

uint32_t L
UIDL.

2.47 Smart Card

FSL_SMARTCARD_DRIVER_VERSION
Smart card driver version 2.3.2.

Smart card Error codes.
Values:

enumerator kStatus_SMARTCARD_Success
Transfer ends successfully

enumerator kStatus_SMARTCARD_TxBusy
Transmit in progress

enumerator kStatus_SMARTCARD_RxBusy
Receiving in progress

enumerator kStatus_SMARTCARD_NoTransferInProgress
No transfer in progress

enumerator kStatus_SMARTCARD_Timeout
Transfer ends with time-out

enumerator kStatus_SMARTCARD_Initialized
Smart card driver is already initialized

enumerator kStatus_SMARTCARD_PhyInitialized
Smart card PHY drive is already initialized

enumerator kStatus_SMARTCARD_CardNotActivated
Smart card is not activated

enumerator kStatus_SMARTCARD_InvalidInput
Function called with invalid input arguments

enumerator kStatus_SMARTCARD_OtherError
Some other error occur

enum _smartcard_control
Control codes for the Smart card protocol timers and misc.
Values:

enumerator kSMARTCARD_EnableADT

enumerator kSMARTCARD_DisableADT

enumerator kSMARTCARD_EnableGTV

enumerator kSMARTCARD_DisableGTV

enumerator kSMARTCARD_ResetWWT

enumerator kSMARTCARD_EnableWWT

enumerator kSMARTCARD_DisableWWT

enumerator kSMARTCARD_ResetCWT

enumerator kSMARTCARD_EnableCWT

enumerator kSMARTCARD_DisableCWT

enumerator kSMARTCARD_ResetBWT

enumerator kSMARTCARD_EnableBWT

enumerator kSMARTCARD_DisableBWT

enumerator kSMARTCARD_EnableInitDetect

enumerator kSMARTCARD_EnableAnack

enumerator kSMARTCARD_DisableAnack

enumerator kSMARTCARD_ConfigureBaudrate

enumerator kSMARTCARD_SetupATRMode

enumerator kSMARTCARD_SetupT0Mode

enumerator kSMARTCARD_SetupT1Mode

enumerator kSMARTCARD_EnableReceiverMode

enumerator kSMARTCARD_DisableReceiverMode

enumerator kSMARTCARD_EnableTransmitterMode

enumerator kSMARTCARD_DisableTransmitterMode

enumerator kSMARTCARD_ResetWaitTimeMultiplier

enum _smartcard_card_voltage_class

Defines Smart card interface voltage class values.

Values:

enumerator kSMARTCARD_VoltageClassUnknown

enumerator kSMARTCARD_VoltageClassA5_0V

enumerator kSMARTCARD_VoltageClassB3_3V

enumerator kSMARTCARD_VoltageClassC1_8V

enum _smartcard_transfer_state

Defines Smart card I/O transfer states.

Values:

enumerator kSMARTCARD_IdleState

enumerator kSMARTCARD_WaitingForTSState

enumerator kSMARTCARD_InvalidTSDetecetedState

enumerator kSMARTCARD_ReceivingState

enumerator kSMARTCARD_TransmittingState

enum _smartcard_reset_type

Defines Smart card reset types.

Values:

enumerator kSMARTCARD_ColdReset

enumerator kSMARTCARD_WarmReset

enumerator kSMARTCARD_NoColdReset

enumerator kSMARTCARD_NoWarmReset

enum _smartcard_transport_type

Defines Smart card transport protocol types.

Values:

enumerator kSMARTCARD_T0Transport

enumerator kSMARTCARD_T1Transport

enum _smartcard_parity_type

Defines Smart card data parity types.

Values:

enumerator kSMARTCARD_EvenParity

enumerator kSMARTCARD_OddParity

enum _smartcard_card_convention

Defines data Convention format.

Values:

enumerator kSMARTCARD_DirectConvention

enumerator kSMARTCARD_InverseConvention

enum _smartcard_interface_control

Defines Smart card interface IC control types.

Values:

enumerator kSMARTCARD_InterfaceSetVcc

enumerator kSMARTCARD_InterfaceSetClockToResetDelay

enumerator kSMARTCARD_InterfaceReadStatus

enum _smartcard_direction

Defines transfer direction.

Values:

enumerator kSMARTCARD_Receive

enumerator kSMARTCARD_Transmit

typedef enum *_smartcard_control* smartcard_control_t
Control codes for the Smart card protocol timers and misc.

typedef enum *_smartcard_card_voltage_class* smartcard_card_voltage_class_t
Defines Smart card interface voltage class values.

typedef enum *_smartcard_transfer_state* smartcard_transfer_state_t
Defines Smart card I/O transfer states.

typedef enum *_smartcard_reset_type* smartcard_reset_type_t
Defines Smart card reset types.

typedef enum *_smartcard_transport_type* smartcard_transport_type_t
Defines Smart card transport protocol types.

typedef enum *_smartcard_parity_type* smartcard_parity_type_t
Defines Smart card data parity types.

typedef enum *_smartcard_card_convention* smartcard_card_convention_t
Defines data Convention format.

typedef enum *_smartcard_interface_control* smartcard_interface_control_t
Defines Smart card interface IC control types.

typedef enum *_smartcard_direction* smartcard_direction_t
Defines transfer direction.

typedef void (*smartcard_interface_callback_t)(void *smartcardContext, void *param)
Smart card interface interrupt callback function type.

typedef void (*smartcard_transfer_callback_t)(void *smartcardContext, void *param)
Smart card transfer interrupt callback function type.

typedef void (*smartcard_time_delay_t)(uint32_t us)
Time Delay function used to passive waiting using RTOS [us].

typedef struct *_smartcard_card_params* smartcard_card_params_t
Defines card-specific parameters for Smart card driver.

typedef struct *_smartcard_timers_state* smartcard_timers_state_t
Smart card defines the state of the EMV timers in the Smart card driver.

typedef struct *_smartcard_interface_config* smartcard_interface_config_t
Defines user specified configuration of Smart card interface.

typedef struct *_smartcard_xfer* smartcard_xfer_t
Defines user transfer structure used to initialize transfer.

typedef struct *_smartcard_context* smartcard_context_t

Runtime state of the Smart card driver.

SMARTCARD_INIT_DELAY_CLOCK_CYCLES

Smart card global define which specify number of clock cycles until initial 'TS' character has to be received.

SMARTCARD_EMV_ATR_DURATION_ETU

Smart card global define which specify number of clock cycles during which ATR string has to be received.

SMARTCARD_TS_DIRECT_CONVENTION

Smart card specification initial TS character definition of direct convention.

SMARTCARD_TS_INVERSE_CONVENTION

Smart card specification initial TS character definition of inverse convention.

struct _smartcard_card_params

#include <fsl_smartcard.h> Defines card-specific parameters for Smart card driver.

Public Members

uint16_t Fi

4 bits Fi - clock rate conversion integer

uint8_t fMax

Maximum Smart card frequency in MHz

uint8_t WI

8 bits WI - work wait time integer

uint8_t Di

4 bits DI - baud rate divisor

uint8_t BWI

4 bits BWI - block wait time integer

uint8_t CWI

4 bits CWI - character wait time integer

uint8_t BGI

BGI - block guard time integer

uint8_t GTN

8 bits GTN - extended guard time integer

uint8_t IFSC

Indicates IFSC value of the card

uint8_t modeNegotiable
: Indicates if the card acts in negotiable or a specific mode.

uint8_t currentD
: 4 bits DI - current baud rate divisor

uint8_t status
: Indicates smart card status

bool t0Indicated
: Indicates ff T=0 indicated in TD1 byte

bool t1Indicated
: Indicates if T=1 indicated in TD2 byte

bool atrComplete
: Indicates whether the ATR received from the card was complete or not

bool atrValid
: Indicates whether the ATR received from the card was valid or not

bool present
: Indicates if a smart card is present

bool active
: Indicates if the smart card is activated

bool faulty
: Indicates whether smart card/interface is faulty

smartcard_card_convention_t convention
: Card convention, kSMARTCARD_DirectConvention for direct convention, kSMARTCARD_InverseConvention for inverse convention

struct _smartcard_timers_state
: *#include <fsl_smartcard.h>* Smart card defines the state of the EMV timers in the Smart card driver.

Public Members

volatile bool adtExpired
: Indicates whether ADT timer expired

volatile bool wwtExpired
: Indicates whether WWT timer expired

volatile bool cwtExpired
: Indicates whether CWT timer expired

volatile bool bwtExpired

Indicates whether BWT timer expired

volatile bool initCharTimerExpired

Indicates whether reception timer for initialization character (TS) after the RST has expired

struct _smartcard_interface_config

#include <fsl_smartcard.h> Defines user specified configuration of Smart card interface.

Public Members

uint32_t smartCardClock

Smart card interface clock [Hz]

uint32_t clockToResetDelay

Indicates clock to RST apply delay [smart card clock cycles]

uint8_t clockModule

Smart card clock module number

uint8_t clockModuleChannel

Smart card clock module channel number

uint8_t clockModuleSourceClock

Smart card clock module source clock [e.g., BusClk]

smartcard_card_voltage_class_t vcc

Smart card voltage class

uint8_t controlPort

Smart card PHY control port instance

uint8_t controlPin

Smart card PHY control pin instance

uint8_t irqPort

Smart card PHY Interrupt port instance

uint8_t irqPin

Smart card PHY Interrupt pin instance

uint8_t resetPort

Smart card reset port instance

uint8_t resetPin

Smart card reset pin instance

uint8_t vsel0Port
: Smart card PHY Vsel0 control port instance

uint8_t vsel0Pin
: Smart card PHY Vsel0 control pin instance

uint8_t vsel1Port
: Smart card PHY Vsel1 control port instance

uint8_t vsel1Pin
: Smart card PHY Vsel1 control pin instance

uint8_t dataPort
: Smart card PHY data port instance

uint8_t dataPin
: Smart card PHY data pin instance

uint8_t dataPinMux
: Smart card PHY data pin mux option

uint8_t tsTimerId
: Numerical identifier of the External HW timer for Initial character detection

struct _smartcard_xfer
: *#include <fsl_smartcard.h>* Defines user transfer structure used to initialize transfer.

Public Members

smartcard_direction_t direction
: Direction of communication. (RX/TX)

uint8_t *buff
: The buffer of data.

size_t size
: The number of transferred units.

struct _smartcard_context
: *#include <fsl_smartcard.h>* Runtime state of the Smart card driver.

Public Members

void *base
: Smart card module base address

smartcard_direction_t direction
: Direction of communication. (RX/TX)

uint8_t *xBuff
: The buffer of data being transferred.

volatile size_t xSize
: The number of bytes to be transferred.

volatile bool xIsBusy
: True if there is an active transfer.

uint8_t txFifoEntryCount
: Number of data word entries in transmit FIFO.

uint8_t rxFifoThreshold
: The max value of the receiver FIFO threshold.

smartcard_interface_callback_t interfaceCallback
: Callback to invoke after interface IC raised interrupt.

smartcard_transfer_callback_t transferCallback
: Callback to invoke after transfer event occur.

void *interfaceCallbackParam
: Interface callback parameter pointer.

void *transferCallbackParam
: Transfer callback parameter pointer.

smartcard_time_delay_t timeDelay
: Function which handles time delay defined by user or RTOS.

smartcard_reset_type_t resetType
: Indicates whether a Cold reset or Warm reset was requested.

smartcard_transport_type_t tType
: Indicates current transfer protocol (T0 or T1)

volatile *smartcard_transfer_state_t* transferState
: Indicates the current transfer state

smartcard_timers_state_t timersState
: Indicates the state of different protocol timers used in driver

smartcard_card_params_t cardParams
: Smart card parameters(ATR and current) and interface slots states(ATR and current)

uint8_t IFSD
: Indicates the terminal IFSD

smartcard_parity_type_t parity
: Indicates current parity even/odd

volatile bool rxtCrossed
: Indicates whether RXT thresholds has been crossed

volatile bool txtCrossed
: Indicates whether TXT thresholds has been crossed

volatile bool wtxRequested
: Indicates whether WTX has been requested or not

volatile bool parityError
: Indicates whether a parity error has been detected

uint8_t statusBytes[2]
: Used to store Status bytes SW1, SW2 of the last executed card command response

smartcard_interface_config_t interfaceConfig
: Smart card interface configuration structure

bool abortTransfer
: Used to abort transfer.

2.48 Smart Card PHY Driver

void SMARTCARD_PHY_GetDefaultConfig(*smartcard_interface_config_t* *config)
: Fills in the configuration structure with default values.

Parameters

- config – The Smart card user configuration structure which contains configuration structure of type smartcard_interface_config_t. Function fill in members: clockToResetDelay = 42000, vcc = kSmartcardVoltageClassB3_3V, with default values.

status_t SMARTCARD_PHY_Init(void *base, *smartcard_interface_config_t* const *config, uint32_t srcClock_Hz)
: Initializes a Smart card interface instance.

Parameters

- base – The Smart card peripheral base address.
- config – The user configuration structure of type smartcard_interface_config_t. Call the function *SMARTCARD_PHY_GetDefaultConfig()* to fill the configuration structure.
- srcClock_Hz – Smart card clock generation module source clock.

Return values

kStatus_SMARTCARD_Success – or kStatus_SMARTCARD_OtherError in case of error.

void SMARTCARD_PHY_Deinit(void *base, *smartcard_interface_config_t* const *config)

De-initializes a Smart card interface, stops the Smart card clock, and disables the VCC.

Parameters

- base – The Smart card peripheral module base address.
- config – The user configuration structure of type smartcard_interface_config_t.

status_t SMARTCARD_PHY_Activate(void *base, *smartcard_context_t* *context, *smartcard_reset_type_t* resetType)

Activates the Smart card IC.

Parameters

- base – The Smart card peripheral module base address.
- context – A pointer to a Smart card driver context structure.
- resetType – type of reset to be performed, possible values = kSmartcardColdReset, kSmartcardWarmReset

Return values

kStatus_SMARTCARD_Success – or kStatus_SMARTCARD_OtherError in case of error.

status_t SMARTCARD_PHY_Deactivate(void *base, *smartcard_context_t* *context)

De-activates the Smart card IC.

Parameters

- base – The Smart card peripheral module base address.
- context – A pointer to a Smart card driver context structure.

Return values

kStatus_SMARTCARD_Success – or kStatus_SMARTCARD_OtherError in case of error.

status_t SMARTCARD_PHY_Control(void *base, *smartcard_context_t* *context, *smartcard_interface_control_t* control, uint32_t param)

Controls the Smart card interface IC.

Parameters

- base – The Smart card peripheral module base address.
- context – A pointer to a Smart card driver context structure.
- control – A interface command type.
- param – Integer value specific to control type

Return values

kStatus_SMARTCARD_Success – or kStatus_SMARTCARD_OtherError in case of error.

SMARTCARD_ATR_DURATION_ADJUSTMENT

Smart card definition which specifies the adjustment number of clock cycles during which an ATR string has to be received.

SMARTCARD_INIT_DELAY_CLOCK_CYCLES_ADJUSTMENT

Smart card definition which specifies the adjustment number of clock cycles until an initial 'TS' character has to be received.

2.49 Smart Card PHY GPIO Driver

2.50 Smart Card PHY TDA8035 Driver

2.51 Smart Card UART Driver

void SMARTCARD_UART_GetDefaultConfig(*smartcard_card_params_t* *cardParams)

Fills in the smartcard_card_params structure with default values according to the EMV 4.3 specification.

Parameters

- cardParams – The configuration structure of type smartcard_interface_config_t. Function fill in members: Fi = 372; Di = 1; currentD = 1; WI = 0x0A; GTN = 0x00; with default values.

status_t SMARTCARD_UART_Init(UART_Type *base, *smartcard_context_t* *context, uint32_t srcClock_Hz)

Initializes a UART peripheral for the Smart card/ISO-7816 operation.

This function un-gates the UART clock, initializes the module to EMV default settings, configures the IRQ, enables the module-level interrupt to the core, and initializes the driver context.

Parameters

- base – The UART peripheral base address.
- context – A pointer to a smart card driver context structure.
- srcClock_Hz – Smart card clock generation module source clock.

Returns

An error code or kStatus_SMARTCARD_Success.

void SMARTCARD_UART_Deinit(UART_Type *base)

This function disables the UART interrupts, disables the transmitter and receiver, and flushes the FIFOs (for modules that support FIFOs) and gates UART clock in SIM.

Parameters

- base – The UART peripheral base address.

int32_t SMARTCARD_UART_GetTransferRemainingBytes(UART_Type *base, *smartcard_context_t* *context)

Returns whether the previous UART transfer has finished.

When performing an async transfer, call this function to ascertain the context of the current transfer: in progress (or busy) or complete (success). If the transfer is still in progress, the user can obtain the number of words that have not been transferred by reading xSize of smart card context structure.

Parameters

- base – The UART peripheral base address.

- context – A pointer to a Smart card driver context structure.

Returns
The number of bytes not transferred.

status_t SMARTCARD_UART_AbortTransfer(UART_Type *base, *smartcard_context_t* *context)

Terminates an asynchronous UART transfer early.

During an async UART transfer, the user can terminate the transfer early if the transfer is still in progress.

Parameters

- base – The UART peripheral base address.
- context – A pointer to a Smart card driver context structure.

Return values

- kStatus_SMARTCARD_Success – The transfer abort was successful.
- kStatus_SMARTCARD_NoTransmitInProgress – No transmission is currently in progress.

status_t SMARTCARD_UART_TransferNonBlocking(UART_Type *base, *smartcard_context_t* *context, *smartcard_xfer_t* *xfer)

Transfers data using interrupts.

A non-blocking (also known as asynchronous) function means that the function returns immediately after initiating the transfer function. The application has to get the transfer status to see when the transfer is complete. In other words, after calling non-blocking (asynchronous) transfer function, the application must get the transfer status to check if transmit is completed or not.

Parameters

- base – The UART peripheral base address.
- context – A pointer to a Smart card driver context structure.
- xfer – A pointer to Smart card transfer structure where the linked buffers and sizes are stored.

Returns
An error code or kStatus_SMARTCARD_Success.

status_t SMARTCARD_UART_Control(UART_Type *base, *smartcard_context_t* *context, *smartcard_control_t* control, uint32_t param)

Controls the UART module per different user requests.

return An kStatus_SMARTCARD_OtherError in case of error return kStatus_SMARTCARD_Success in success

Parameters

- base – The UART peripheral base address.
- context – A pointer to a smart card driver context structure.
- control – Smart card command type.
- param – Integer value specific to a control command.

void SMARTCARD_UART_IRQHandler(UART_Type *base, *smartcard_context_t* *context)

Interrupt handler for UART.

This handler uses the buffers stored in the smartcard_context_t structures to transfer data. The Smart card driver requires this function to call when the UART interrupt occurs.

Parameters

- base – The UART peripheral base address.
- context – A pointer to a Smart card driver context structure.

void SMARTCARD_UART_ErrIRQHandler(UART_Type *base, *smartcard_context_t* *context)

Error interrupt handler for UART.

This function handles error conditions during a transfer.

Parameters

- base – The UART peripheral base address.
- context – A pointer to a Smart card driver context structure.

void SMARTCARD_UART_TSExpiryCallback(UART_Type *base, *smartcard_context_t* *context)

Handles initial TS character timer time-out event.

Parameters

- base – The UART peripheral base address.
- context – A pointer to a Smart card driver context structure.

void smartcard_uart_TimerStart(uint8_t channel, uint32_t time)

Initializes timer specific channel with input period, enable channel interrupt and start counter.

Parameters

- channel – The timer channel.
- time – The time period.

SMARTCARD_EMV_RX_NACK_THRESHOLD

EMV RX NACK interrupt generation threshold.

SMARTCARD_EMV_TX_NACK_THRESHOLD

EMV TX NACK interrupt generation threshold.

SMARTCARD_EMV_RX_TO_TX_GUARD_TIME_T0

EMV TX & RX GUART TIME default value.

SBR_CAL_ADJUST_D1_T0

BRFA_CAL_ADJUST_D1_T0

SBR_CAL_ADJUST_D2_T0

BRFA_CAL_ADJUST_D2_T0

SBR_CAL_ADJUST_D4_T0

BRFA_CAL_ADJUST_D4_T0

SBR_CAL_ADJUST_D1_T1

BRFA_CAL_ADJUST_D1_T1

SBR_CAL_ADJUST_D2_T1

BRFA_CAL_ADJUST_D2_T1

SBR_CAL_ADJUST_D4_T1

BRFA_CAL_ADJUST_D4_T1

2.52 SMC: System Mode Controller Driver

static inline void SMC_GetVersionId(SMC_Type *base, *smc_version_id_t* *versionId)

Gets the SMC version ID.

This function gets the SMC version ID, including major version number, minor version number, and feature specification number.

Parameters

- base – SMC peripheral base address.
- versionId – Pointer to the version ID structure.

void SMC_GetParam(SMC_Type *base, *smc_param_t* *param)

Gets the SMC parameter.

This function gets the SMC parameter including the enabled power mdoes.

Parameters

- base – SMC peripheral base address.
- param – Pointer to the SMC param structure.

static inline void SMC_SetPowerModeProtection(SMC_Type *base, uint8_t allowedModes)

Configures all power mode protection settings.

This function configures the power mode protection settings for supported power modes in the specified chip family. The available power modes are defined in the smc_power_mode_protection_t. This should be done at an early system level initialization stage. See the reference manual for details. This register can only write once after the power reset.

The allowed modes are passed as bit map. For example, to allow LLS and VLLS, use SMC_SetPowerModeProtection(kSMC_AllowPowerModeVlls | kSMC_AllowPowerModeVlps). To allow all modes, use SMC_SetPowerModeProtection(kSMC_AllowPowerModeAll).

Parameters

- base – SMC peripheral base address.
- allowedModes – Bitmap of the allowed power modes.

static inline *smc_power_state_t* SMC_GetPowerModeState(SMC_Type *base)

Gets the current power mode status.

This function returns the current power mode status. After the application switches the power mode, it should always check the status to check whether it runs into the specified

mode or not. The application should check this mode before switching to a different mode. The system requires that only certain modes can switch to other specific modes. See the reference manual for details and the smc_power_state_t for information about the power status.

Parameters

- base – SMC peripheral base address.

Returns

Current power mode status.

void SMC_PreEnterStopModes(void)

Prepares to enter stop modes.

This function should be called before entering STOP/VLPS/LLS/VLLS modes.

void SMC_PostExitStopModes(void)

Recovers after wake up from stop modes.

This function should be called after wake up from STOP/VLPS/LLS/VLLS modes. It is used with *SMC_PreEnterStopModes*.

void SMC_PreEnterWaitModes(void)

Prepares to enter wait modes.

This function should be called before entering WAIT/VLPW modes.

void SMC_PostExitWaitModes(void)

Recovers after wake up from stop modes.

This function should be called after wake up from WAIT/VLPW modes. It is used with *SMC_PreEnterWaitModes*.

status_t SMC_SetPowerModeRun(SMC_Type *base)

Configures the system to RUN power mode.

Parameters

- base – SMC peripheral base address.

Returns

SMC configuration error code.

status_t SMC_SetPowerModeHsrun(SMC_Type *base)

Configures the system to HSRUN power mode.

Parameters

- base – SMC peripheral base address.

Returns

SMC configuration error code.

status_t SMC_SetPowerModeWait(SMC_Type *base)

Configures the system to WAIT power mode.

Parameters

- base – SMC peripheral base address.

Returns

SMC configuration error code.

status_t SMC_SetPowerModeStop(SMC_Type *base, *smc_partial_stop_option_t* option)

Configures the system to Stop power mode.

Parameters

- base – SMC peripheral base address.
- option – Partial Stop mode option.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeVlpr(SMC_Type *base, bool wakeupMode)

Configures the system to VLPR power mode.

Parameters

- base – SMC peripheral base address.
- wakeupMode – Enter Normal Run mode if true, else stay in VLPR mode.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeVlpw(SMC_Type *base)

Configures the system to VLPW power mode.

Parameters

- base – SMC peripheral base address.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeVlps(SMC_Type *base)

Configures the system to VLPS power mode.

Parameters

- base – SMC peripheral base address.

Returns
SMC configuration error code.

status_t SMC_SetPowerModeLls(SMC_Type *base, const *smc_power_mode_lls_config_t* *config)

Configures the system to LLS power mode.

Parameters

- base – SMC peripheral base address.
- config – The LLS power mode configuration structure

Returns
SMC configuration error code.

status_t SMC_SetPowerModeVlls(SMC_Type *base, const *smc_power_mode_vlls_config_t* *config)

Configures the system to VLLS power mode.

Parameters

- base – SMC peripheral base address.
- config – The VLLS power mode configuration structure.

Returns
SMC configuration error code.

FSL_SMC_DRIVER_VERSION

SMC driver version.

enum _smc_power_mode_protection

Power Modes Protection.

Values:

enumerator kSMC_AllowPowerModeVlls

Allow Very-low-leakage Stop Mode.

enumerator kSMC_AllowPowerModeLls

Allow Low-leakage Stop Mode.

enumerator kSMC_AllowPowerModeVlp

Allow Very-Low-power Mode.

enumerator kSMC_AllowPowerModeHsrun

Allow High-speed Run mode.

enumerator kSMC_AllowPowerModeAll

Allow all power mode.

enum _smc_power_state

Power Modes in PMSTAT.

Values:

enumerator kSMC_PowerStateRun

0000_0001 - Current power mode is RUN

enumerator kSMC_PowerStateStop

0000_0010 - Current power mode is STOP

enumerator kSMC_PowerStateVlpr

0000_0100 - Current power mode is VLPR

enumerator kSMC_PowerStateVlpw

0000_1000 - Current power mode is VLPW

enumerator kSMC_PowerStateVlps

0001_0000 - Current power mode is VLPS

enumerator kSMC_PowerStateLls

0010_0000 - Current power mode is LLS

enumerator kSMC_PowerStateVlls

0100_0000 - Current power mode is VLLS

enumerator kSMC_PowerStateHsrun

1000_0000 - Current power mode is HSRUN

enum _smc_run_mode

Run mode definition.

Values:

enumerator kSMC_RunNormal

Normal RUN mode.

enumerator kSMC_RunVlpr

Very-low-power RUN mode.

enumerator kSMC_Hsrun

High-speed Run mode (HSRUN).

enum _smc_stop_mode

Stop mode definition.

Values:

enumerator kSMC_StopNormal

Normal STOP mode.

enumerator kSMC_StopVlps

Very-low-power STOP mode.

enumerator kSMC_StopLls

Low-leakage Stop mode.

enumerator kSMC_StopVlls

Very-low-leakage Stop mode.

enum _smc_stop_submode

VLLS/LLS stop sub mode definition.

Values:

enumerator kSMC_StopSub0

Stop submode 0, for VLLS0/LLS0.

enumerator kSMC_StopSub1

Stop submode 1, for VLLS1/LLS1.

enumerator kSMC_StopSub2

Stop submode 2, for VLLS2/LLS2.

enumerator kSMC_StopSub3

Stop submode 3, for VLLS3/LLS3.

enum _smc_partial_stop_mode

Partial STOP option.

Values:

enumerator kSMC_PartialStop
STOP - Normal Stop mode

enumerator kSMC_PartialStop1
Partial Stop with both system and bus clocks disabled

enumerator kSMC_PartialStop2
Partial Stop with system clock disabled and bus clock enabled

_smc_status, SMC configuration status.
Values:

enumerator kStatus_SMC_StopAbort
Entering Stop mode is abort

typedef enum *_smc_power_mode_protection* smc_power_mode_protection_t
Power Modes Protection.

typedef enum *_smc_power_state* smc_power_state_t
Power Modes in PMSTAT.

typedef enum *_smc_run_mode* smc_run_mode_t
Run mode definition.

typedef enum *_smc_stop_mode* smc_stop_mode_t
Stop mode definition.

typedef enum *_smc_stop_submode* smc_stop_submode_t
VLLS/LLS stop sub mode definition.

typedef enum *_smc_partial_stop_mode* smc_partial_stop_option_t
Partial STOP option.

typedef struct *_smc_version_id* smc_version_id_t
IP version ID definition.

typedef struct *_smc_param* smc_param_t
IP parameter definition.

typedef struct *_smc_power_mode_lls_config* smc_power_mode_lls_config_t
SMC Low-Leakage Stop power mode configuration.

typedef struct *_smc_power_mode_vlls_config* smc_power_mode_vlls_config_t
SMC Very Low-Leakage Stop power mode configuration.

struct _smc_version_id
#include <fsl_smc.h> IP version ID definition.

Public Members

uint16_t feature
: Feature Specification Number.

uint8_t minor
: Minor version number.

uint8_t major
: Major version number.

struct __smc__param
: *#include <fsl_smc.h>* IP parameter definition.

Public Members

bool hsrunEnable
: HSRUN mode enable.

bool llsEnable
: LLS mode enable.

bool lls2Enable
: LLS2 mode enable.

bool vlls0Enable
: VLLS0 mode enable.

struct __smc__power__mode__lls__config
: *#include <fsl_smc.h>* SMC Low-Leakage Stop power mode configuration.

Public Members

smc_stop_submode_t subMode
: Low-leakage Stop sub-mode

bool enableLpoClock
: Enable LPO clock in LLS mode

struct __smc__power__mode__vlls__config
: *#include <fsl_smc.h>* SMC Very Low-Leakage Stop power mode configuration.

Public Members

smc_stop_submode_t subMode
: Very Low-leakage Stop sub-mode

bool enablePorDetectInVlls0

Enable Power on reset detect in VLLS mode

bool enableRam2InVlls2

Enable RAM2 power in VLLS2

bool enableLpoClock

Enable LPO clock in VLLS mode

2.53 UART: Universal Asynchronous Receiver/Transmitter Driver

2.54 UART Driver

status_t UART_Init(UART_Type *base, const *uart_config_t* *config, uint32_t srcClock_Hz)

Initializes a UART instance with a user configuration structure and peripheral clock.

This function configures the UART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the *UART_GetDefaultConfig()* function. The example below shows how to use this API to configure UART.

```
uart_config_t uartConfig;
uartConfig.baudRate_Bps = 115200U;
uartConfig.parityMode = kUART_ParityDisabled;
uartConfig.stopBitCount = kUART_OneStopBit;
uartConfig.txFifoWatermark = 0;
uartConfig.rxFifoWatermark = 1;
UART_Init(UART1, &uartConfig, 20000000U);
```

Parameters

- base – UART peripheral base address.
- config – Pointer to the user-defined configuration structure.
- srcClock_Hz – UART clock source frequency in HZ.

Return values

- kStatus_UART_BaudrateNotSupport – Baudrate is not support in current clock source.
- kStatus_Success – Status UART initialize succeed

void UART_Deinit(UART_Type *base)

Deinitializes a UART instance.

This function waits for TX complete, disables TX and RX, and disables the UART clock.

Parameters

- base – UART peripheral base address.

void UART_GetDefaultConfig(*uart_config_t* *config)

Gets the default configuration structure.

This function initializes the UART configuration structure to a default value. The default values are as follows. uartConfig->baudRate_Bps = 115200U; uartConfig->bitCountPerChar = kUART_8BitsPerChar; uartConfig->parityMode = kUART_ParityDisabled; uartConfig->stopBitCount = kUART_OneStopBit; uartConfig->txFifoWatermark = 0; uartConfig->rxFifoWatermark = 1; uartConfig->idleType = kUART_IdleTypeStartBit; uartConfig->enableTx = false; uartConfig->enableRx = false;

Parameters

- config – Pointer to configuration structure.

status_t UART_SetBaudRate(UART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)

Sets the UART instance baud rate.

This function configures the UART module baud rate. This function is used to update the UART module baud rate after the UART module is initialized by the UART_Init.

```
UART_SetBaudRate(UART1, 115200U, 20000000U);
```

Parameters

- base – UART peripheral base address.
- baudRate_Bps – UART baudrate to be set.
- srcClock_Hz – UART clock source frequency in Hz.

Return values

- kStatus_UART_BaudrateNotSupport – Baudrate is not support in the current clock source.
- kStatus_Success – Set baudrate succeeded.

void UART_Enable9bitMode(UART_Type *base, bool enable)

Enable 9-bit data mode for UART.

This function set the 9-bit mode for UART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

- base – UART peripheral base address.
- enable – true to enable, flase to disable.

static inline void UART_SetMatchAddress(UART_Type *base, uint8_t address1, uint8_t address2)

Set the UART slave address.

This function configures the address for UART module that works as slave in 9-bit data mode. One or two address fields can be configured. When the address field's match enable bit is set, the frame it receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches one of slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note: Any UART 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 – UART peripheral base address.
- address1 – UART slave address 1.
- address2 – UART slave address 2.

static inline void UART_EnableMatchAddress(UART_Type *base, bool match1, bool match2)

Enable the UART match address feature.

Parameters

- base – UART peripheral base address.
- match1 – true to enable match address1, false to disable.
- match2 – true to enable match address2, false to disable.

static inline void UART_Set9thTransmitBit(UART_Type *base)

Set UART 9th transmit bit.

Parameters

- base – UART peripheral base address.

static inline void UART_Clear9thTransmitBit(UART_Type *base)

Clear UART 9th transmit bit.

Parameters

- base – UART peripheral base address.

uint32_t UART_GetStatusFlags(UART_Type *base)

Gets UART status flags.

This function gets all UART status flags. The flags are returned as the logical OR value of the enumerators _uart_flags. To check a specific status, compare the return value with enumerators in _uart_flags. For example, to check whether the TX is empty, do the following.

```
if (kUART_TxDataRegEmptyFlag & UART_GetStatusFlags(UART1))
{
    ...
}
```

Parameters

- base – UART peripheral base address.

Returns

UART status flags which are ORed by the enumerators in the _uart_flags.

status_t UART_ClearStatusFlags(UART_Type *base, uint32_t mask)

Clears status flags with the provided mask.

This function clears UART status flags with a provided mask. An automatically cleared flag can't be cleared by this function. These flags can only be cleared or set by hardware. kUART_TxDataRegEmptyFlag, kUART_TransmissionCompleteFlag, kUART_RxDataRegFullFlag, kUART_RxActiveFlag, kUART_NoiseErrorInRxDataRegFlag, kUART_ParityErrorInRxDataRegFlag, kUART_TxFifoEmptyFlag,kUART_RxFifoEmptyFlag

Note: that this API should be called when the Tx/Rx is idle. Otherwise it has no effect.

Parameters

- base – UART peripheral base address.
- mask – The status flags to be cleared; it is logical OR value of _uart_flags.

Return values

- kStatus_UART_FlagCannotClearManually – The flag can't be cleared by this function but it is cleared automatically by hardware.
- kStatus_Success – Status in the mask is cleared.

void UART_EnableInterrupts(UART_Type *base, uint32_t mask)

Enables UART interrupts according to the provided mask.

This function enables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See *_uart_interrupt_enable*. For example, to enable TX empty interrupt and RX full interrupt, do the following.

```
UART_EnableInterrupts(UART1,kUART_TxDataRegEmptyInterruptEnable | kUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – UART peripheral base address.
- mask – The interrupts to enable. Logical OR of *_uart_interrupt_enable*.

void UART_DisableInterrupts(UART_Type *base, uint32_t mask)

Disables the UART interrupts according to the provided mask.

This function disables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See *_uart_interrupt_enable*. For example, to disable TX empty interrupt and RX full interrupt do the following.

```
UART_DisableInterrupts(UART1,kUART_TxDataRegEmptyInterruptEnable | kUART_
↪RxDataRegFullInterruptEnable);
```

Parameters

- base – UART peripheral base address.
- mask – The interrupts to disable. Logical OR of *_uart_interrupt_enable*.

uint32_t UART_GetEnabledInterrupts(UART_Type *base)

Gets the enabled UART interrupts.

This function gets the enabled UART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators *_uart_interrupt_enable*. To check a specific interrupts enable status, compare the return value with enumerators in *_uart_interrupt_enable*. For example, to check whether TX empty interrupt is enabled, do the following.

```
uint32_t enabledInterrupts = UART_GetEnabledInterrupts(UART1);

if (kUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
{
    ...
}
```

Parameters

- base – UART peripheral base address.

Returns

UART interrupt flags which are logical OR of the enumerators in *_uart_interrupt_enable*.

static inline uint32_t UART_GetDataRegisterAddress(UART_Type *base)

Gets the UART data register address.

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

- base – UART peripheral base address.

Returns

UART data register addresses which are used both by the transmitter and the receiver.

static inline void UART_EnableTxDMA(UART_Type *base, bool enable)

Enables or disables the UART transmitter DMA request.

This function enables or disables the transmit data register empty flag, S1[TDRE], to generate the DMA requests.

Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

static inline void UART_EnableRxDMA(UART_Type *base, bool enable)

Enables or disables the UART receiver DMA.

This function enables or disables the receiver data register full flag, S1[RDRF], to generate DMA requests.

Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

static inline void UART_EnableTx(UART_Type *base, bool enable)

Enables or disables the UART transmitter.

This function enables or disables the UART transmitter.

Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

static inline void UART_EnableRx(UART_Type *base, bool enable)

Enables or disables the UART receiver.

This function enables or disables the UART receiver.

Parameters

- base – UART peripheral base address.
- enable – True to enable, false to disable.

static inline void UART_WriteByte(UART_Type *base, uint8_t data)

Writes to the TX register.

This function writes data to the TX register directly. The upper layer must ensure that the TX register is empty or TX FIFO has empty room before calling this function.

Parameters

- base – UART peripheral base address.
- data – The byte to write.

static inline uint8_t UART_ReadByte(UART_Type *base)

Reads the RX register directly.

This function reads data from the RX register directly. The upper layer must ensure that the RX register is full or that the TX FIFO has data before calling this function.

Parameters

- base – UART peripheral base address.

Returns

The byte read from UART data register.

static inline uint8_t UART_GetRxFifoCount(UART_Type *base)

Gets the rx FIFO data count.

Parameters

- base – UART peripheral base address.

Returns

rx FIFO data count.

static inline uint8_t UART_GetTxFifoCount(UART_Type *base)

Gets the tx FIFO data count.

Parameters

- base – UART peripheral base address.

Returns

tx FIFO data count.

void UART_SendAddress(UART_Type *base, uint8_t address)

Transmit an address frame in 9-bit data mode.

Parameters

- base – UART peripheral base address.
- address – UART slave address.

status_t UART_WriteBlocking(UART_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 – UART peripheral base address.
- data – Start address of the data to write.
- length – Size of the data to write.

Return values

- kStatus_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully wrote all data.

status_t UART_ReadBlocking(UART_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 reads data from the TX register.

Parameters

- base – UART peripheral base address.
- data – Start address of the buffer to store the received data.
- length – Size of the buffer.

Return values

- kStatus_UART_RxHardwareOverrun – Receiver overrun occurred while receiving data.
- kStatus_UART_NoiseError – A noise error occurred while receiving data.
- kStatus_UART_FramingError – A framing error occurred while receiving data.
- kStatus_UART_ParityError – A parity error occurred while receiving data.
- kStatus_UART_Timeout – Transmission timed out and was aborted.
- kStatus_Success – Successfully received all data.

void UART_TransferCreateHandle(UART_Type *base, *uart_handle_t* *handle, *uart_transfer_callback_t* callback, void *userData)

Initializes the UART handle.

This function initializes the UART handle which can be used for other UART transactional APIs. Usually, for a specified UART instance, call this API once to get the initialized handle.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- callback – The callback function.
- userData – The parameter of the callback function.

void UART_TransferStartRingBuffer(UART_Type *base, *uart_handle_t* *handle, uint8_t *ringBuffer, size_t ringBufferSize)

Sets up the RX ring buffer.

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the *UART_TransferReceiveNonBlocking()* API. If data is already 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, only 31 bytes are used for saving data.

Parameters

- base – UART peripheral base address.
- handle – UART 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 UART_TransferStopRingBuffer(UART_Type *base, *uart_handle_t* *handle)

Aborts the background transfer and uninstalls the ring buffer.

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.

size_t UART_TransferGetRxRingBufferLength(*uart_handle_t* *handle)

Get the length of received data in RX ring buffer.

Parameters

- handle – UART handle pointer.

Returns

Length of received data in RX ring buffer.

status_t UART_TransferSendNonBlocking(UART_Type *base, *uart_handle_t* *handle, *uart_transfer_t* *xfer)

Transmits a buffer of data using the interrupt method.

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the ISR, the UART driver calls the callback function and passes the *kStatus_UART_TxIdle* as status parameter.

Note: The kStatus_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out. Before disabling the TX, check the kUART_TransmissionCompleteFlag to ensure that the TX is finished.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- xfer – UART transfer structure. See *uart_transfer_t*.

Return values

- kStatus_Success – Successfully start the data transmission.
- kStatus_UART_TxBusy – Previous transmission still not finished; data not all written to TX register yet.
- kStatus_InvalidArgument – Invalid argument.

void UART_TransferAbortSend(UART_Type *base, *uart_handle_t* *handle)

Aborts the interrupt-driven data transmit.

This function aborts the interrupt-driven data sending. The user can get the remainBytes to find out how many bytes are not sent out.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.

status_t UART_TransferGetSendCount(UART_Type *base, *uart_handle_t* *handle, uint32_t *count)

Gets the number of bytes sent out to bus.

This function gets the number of bytes sent out to bus by using the interrupt method.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- count – Send bytes count.

Return values

- kStatus_NoTransferInProgress – No send in progress.
- kStatus_InvalidArgument – The parameter is invalid.
- kStatus_Success – Get successfully through the parameter count;

status_t UART_TransferReceiveNonBlocking(UART_Type *base, *uart_handle_t* *handle, *uart_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 UART driver. When the new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter *kStatus_UART_RxIdle*. For example, 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 UART 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 – UART peripheral base address.
- handle – UART handle pointer.
- xfer – UART transfer structure, see *uart_transfer_t*.
- receivedBytes – Bytes received from the ring buffer directly.

Return values

- kStatus_Success – Successfully queue the transfer into transmit queue.
- kStatus_UART_RxBusy – Previous receive request is not finished.
- kStatus_InvalidArgument – Invalid argument.

void UART_TransferAbortReceive(UART_Type *base, *uart_handle_t* *handle)

Aborts the interrupt-driven data receiving.

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to know how many bytes are not received yet.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.

status_t UART_TransferGetReceiveCount(UART_Type *base, *uart_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been received.

This function gets the number of bytes that have been received.

Parameters

- base – UART peripheral base address.
- handle – UART 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 UART_EnableTxFIFO(UART_Type *base, bool enable)

Enables or disables the UART Tx FIFO.

This function enables or disables the UART Tx FIFO.

param base UART peripheral base address. param enable true to enable, false to disable. retval kStatus_Success Successfully turn on or turn off Tx FIFO. retval kStatus_Fail Fail to turn on or turn off Tx FIFO.

status_t UART_EnableRxFIFO(UART_Type *base, bool enable)

Enables or disables the UART Rx FIFO.

This function enables or disables the UART Rx FIFO.

param base UART peripheral base address. param enable true to enable, false to disable. retval kStatus_Success Successfully turn on or turn off Rx FIFO. retval kStatus_Fail Fail to turn on or turn off Rx FIFO.

static inline void UART_SetRxFifoWatermark(UART_Type *base, uint8_t water)

Sets the rx FIFO watermark.

Parameters

- base – UART peripheral base address.
- water – Rx FIFO watermark.

static inline void UART_SetTxFifoWatermark(UART_Type *base, uint8_t water)

Sets the tx FIFO watermark.

Parameters

- base – UART peripheral base address.
- water – Tx FIFO watermark.

void UART_TransferHandleIRQ(UART_Type *base, void *irqHandle)

UART IRQ handle function.

This function handles the UART transmit and receive IRQ request.

Parameters

- base – UART peripheral base address.
- irqHandle – UART handle pointer.

void UART_TransferHandleErrorIRQ(UART_Type *base, void *irqHandle)

UART Error IRQ handle function.

This function handles the UART error IRQ request.

Parameters

- base – UART peripheral base address.
- irqHandle – UART handle pointer.

void UART_DriverIRQHandler(uint32_t instance)

UART common IRQ handler.

This function handles the UART IRQ request. It can be used by new SOCs where each vector table entry calls this common handler with the instance index, instead of defining a separate handler per instance.

Parameters

- instance – UART instance index.

FSL_UART_DRIVER_VERSION

UART driver version.

Error codes for the UART driver.

Values:

enumerator kStatus_UART_TxBusy

Transmitter is busy.

enumerator kStatus_UART_RxBusy

Receiver is busy.

enumerator kStatus_UART_TxIdle

UART transmitter is idle.

enumerator kStatus_UART_RxIdle

UART receiver is idle.

enumerator kStatus_UART_TxWatermarkTooLarge

TX FIFO watermark too large

enumerator kStatus_UART_RxWatermarkTooLarge

RX FIFO watermark too large

enumerator kStatus_UART_FlagCannotClearManually

UART flag can't be manually cleared.

enumerator kStatus_UART_Error

Error happens on UART.

enumerator kStatus_UART_RxRingBufferOverrun

UART RX software ring buffer overrun.

enumerator kStatus_UART_RxHardwareOverrun
UART RX receiver overrun.

enumerator kStatus_UART_NoiseError
UART noise error.

enumerator kStatus_UART_FramingError
UART framing error.

enumerator kStatus_UART_ParityError
UART parity error.

enumerator kStatus_UART_BaudrateNotSupport
Baudrate is not support in current clock source

enumerator kStatus_UART_IdleLineDetected
UART IDLE line detected.

enumerator kStatus_UART_Timeout
UART times out.

enum _uart_parity_mode
UART parity mode.
Values:

enumerator kUART_ParityDisabled
Parity disabled

enumerator kUART_ParityEven
Parity enabled, type even, bit setting: PE|PT = 10

enumerator kUART_ParityOdd
Parity enabled, type odd, bit setting: PE|PT = 11

enum _uart_stop_bit_count
UART stop bit count.
Values:

enumerator kUART_OneStopBit
One stop bit

enumerator kUART_TwoStopBit
Two stop bits

enum _uart_idle_type_select
UART idle type select.
Values:

enumerator kUART_IdleTypeStartBit
Start counting after a valid start bit.

enumerator kUART_IdleTypeStopBit
Start counting after a stop bit.

enum _uart_interrupt_enable
UART interrupt configuration structure, default settings all disabled.
This structure contains the settings for all of the UART interrupt configurations.
Values:

enumerator kUART_LinBreakInterruptEnable
LIN break detect interrupt.

enumerator kUART_RxActiveEdgeInterruptEnable
RX active edge interrupt.

enumerator kUART_TxDataRegEmptyInterruptEnable
Transmit data register empty interrupt.

enumerator kUART_TransmissionCompleteInterruptEnable
Transmission complete interrupt.

enumerator kUART_RxDataRegFullInterruptEnable
Receiver data register full interrupt.

enumerator kUART_IdleLineInterruptEnable
Idle line interrupt.

enumerator kUART_RxOverrunInterruptEnable
Receiver overrun interrupt.

enumerator kUART_NoiseErrorInterruptEnable
Noise error flag interrupt.

enumerator kUART_FramingErrorInterruptEnable
Framing error flag interrupt.

enumerator kUART_ParityErrorInterruptEnable
Parity error flag interrupt.

enumerator kUART_RxFifoOverflowInterruptEnable
RX FIFO overflow interrupt.

enumerator kUART_TxFifoOverflowInterruptEnable
TX FIFO overflow interrupt.

enumerator kUART_RxFifoUnderflowInterruptEnable
RX FIFO underflow interrupt.

enumerator kUART_AllInterruptsEnable

UART status flags.

This provides constants for the UART status flags for use in the UART functions.

Values:

enumerator kUART_TxDataRegEmptyFlag
TX data register empty flag.

enumerator kUART_TransmissionCompleteFlag
Transmission complete flag.

enumerator kUART_RxDataRegFullFlag
RX data register full flag.

enumerator kUART_IdleLineFlag
Idle line detect flag.

enumerator kUART_RxOverrunFlag
RX overrun flag.

enumerator kUART_NoiseErrorFlag
RX takes 3 samples of each received bit. If any of these samples differ, noise flag sets

enumerator kUART_FramingErrorFlag
Frame error flag, sets if logic 0 was detected where stop bit expected

enumerator kUART_ParityErrorFlag
If parity enabled, sets upon parity error detection

enumerator kUART_LinBreakFlag
LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled

enumerator kUART_RxActiveEdgeFlag
RX pin active edge interrupt flag,sets when active edge detected

enumerator kUART_RxActiveFlag
Receiver Active Flag (RAF), sets at beginning of valid start bit

enumerator kUART_NoiseErrorInRxDataRegFlag
Noisy bit, sets if noise detected.

enumerator kUART_ParityErrorInRxDataRegFlag
Parity bit, sets if parity error detected.

enumerator kUART_TxFifoEmptyFlag
TXEMPT bit, sets if TX buffer is empty

enumerator kUART_RxFifoEmptyFlag
RXEMPT bit, sets if RX buffer is empty

enumerator kUART_TxFifoOverflowFlag
TXOF bit, sets if TX buffer overflow occurred

enumerator kUART_RxFifoOverflowFlag
RXOF bit, sets if receive buffer overflow

enumerator kUART_RxFifoUnderflowFlag
RXUF bit, sets if receive buffer underflow

typedef enum *_uart_parity_mode* uart_parity_mode_t
UART parity mode.

typedef enum *_uart_stop_bit_count* uart_stop_bit_count_t
UART stop bit count.

typedef enum *_uart_idle_type_select* uart_idle_type_select_t
UART idle type select.

typedef struct *_uart_config* uart_config_t
UART configuration structure.

typedef struct *_uart_transfer* uart_transfer_t
UART transfer structure.

typedef struct *_uart_handle* uart_handle_t

typedef void (*uart_transfer_callback_t)(UART_Type *base, *uart_handle_t* *handle, *status_t* status, void *userData)
UART transfer callback function.

typedef void (*uart_isr_t)(UART_Type *base, void *handle)

void *s_uartHandle[]
Pointers to uart handles for each instance.

const IRQn_Type s_uartIRQ[]

uart_isr_t s_uartIsr
Pointer to uart IRQ handler for each instance.

uint32_t UART_GetInstance(UART_Type *base)

Get the UART instance from peripheral base address.

Parameters

- base – UART peripheral base address.

Returns

UART instance.

UART_RETRY_TIMES

Retry times for waiting flag.

struct _uart_config

#include <fsl_uart.h> UART configuration structure.

Public Members

uint32_t baudRate_Bps

UART baud rate

uart_parity_mode_t parityMode

Parity mode, disabled (default), even, odd

uart_stop_bit_count_t stopBitCount

Number of stop bits, 1 stop bit (default) or 2 stop bits

uint8_t txFifoWatermark

TX FIFO watermark

uint8_t rxFifoWatermark

RX FIFO watermark

bool enableRxRTS

RX RTS enable

bool enableTxCTS

TX CTS enable

uart_idle_type_select_t idleType

IDLE type select.

bool enableTx

Enable TX

bool enableRx

Enable RX

struct _uart_transfer

#include <fsl_uart.h> UART transfer structure.

Public Members

size_t dataSize
The byte count to be transfer.

struct __uart_handle
#include <fsl_uart.h> UART handle structure.

Public Members

const uint8_t *volatile txData
Address of remaining data to send.

volatile size_t txDataSize
Size of the remaining data to send.

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.

uart_transfer_callback_t callback
Callback function.

void *userData
UART callback function parameter.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

union ___unnamed39___

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.

2.55 UART eDMA Driver

void UART_TransferCreateHandleEDMA(UART_Type *base, *uart_edma_handle_t* *handle, *uart_edma_transfer_callback_t* callback, void *userData, *edma_handle_t* *txEdmaHandle, *edma_handle_t* *rxEdmaHandle)

Initializes the UART handle which is used in transactional functions.

Parameters

- base – UART peripheral base address.
- handle – Pointer to the uart_edma_handle_t structure.
- callback – UART callback, NULL means no callback.
- userData – User callback function data.
- rxEdmaHandle – User-requested DMA handle for RX DMA transfer.
- txEdmaHandle – User-requested DMA handle for TX DMA transfer.

status_t UART_SendEDMA(UART_Type *base, *uart_edma_handle_t* *handle, *uart_transfer_t* *xfer)

Sends data using eDMA.

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

- base – UART peripheral base address.
- handle – UART handle pointer.
- xfer – UART eDMA transfer structure. See *uart_transfer_t*.

Return values

- kStatus_Success – if succeeded; otherwise failed.
- kStatus_UART_TxBusy – Previous transfer ongoing.

- kStatus_InvalidArgument – Invalid argument.

status_t UART_ReceiveEDMA(UART_Type *base, *uart_edma_handle_t* *handle, *uart_transfer_t* *xfer)

Receives data using eDMA.

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

- base – UART peripheral base address.
- handle – Pointer to the uart_edma_handle_t structure.
- xfer – UART eDMA transfer structure. See *uart_transfer_t*.

Return values

- kStatus_Success – if succeeded; otherwise failed.
- kStatus_UART_RxBusy – Previous transfer ongoing.
- kStatus_InvalidArgument – Invalid argument.

void UART_TransferAbortSendEDMA(UART_Type *base, *uart_edma_handle_t* *handle)

Aborts the sent data using eDMA.

This function aborts sent data using eDMA.

Parameters

- base – UART peripheral base address.
- handle – Pointer to the uart_edma_handle_t structure.

void UART_TransferAbortReceiveEDMA(UART_Type *base, *uart_edma_handle_t* *handle)

Aborts the receive data using eDMA.

This function aborts receive data using eDMA.

Parameters

- base – UART peripheral base address.
- handle – Pointer to the uart_edma_handle_t structure.

status_t UART_TransferGetSendCountEDMA(UART_Type *base, *uart_edma_handle_t* *handle, uint32_t *count)

Gets the number of bytes that have been written to UART TX register.

This function gets the number of bytes that have been written to UART TX register by DMA.

Parameters

- base – UART peripheral base address.
- handle – UART 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 UART_TransferGetReceiveCountEDMA(UART_Type *base, *uart_edma_handle_t* *handle, uint32_t *count)

Gets the number of received bytes.

This function gets the number of received bytes.

Parameters

- base – UART peripheral base address.
- handle – UART 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 UART_TransferEdmaHandleIRQ(UART_Type *base, void *uartEdmaHandle)

UART eDMA IRQ handle function.

This function handles the UART transmit complete IRQ request and invoke user callback.

Parameters

- base – UART peripheral base address.
- uartEdmaHandle – UART handle pointer.

FSL_UART_EDMA_DRIVER_VERSION

UART EDMA driver version.

typedef struct *_uart_edma_handle* uart_edma_handle_t

typedef void (*uart_edma_transfer_callback_t)(UART_Type *base, *uart_edma_handle_t* *handle, *status_t* status, void *userData)

UART transfer callback function.

struct _uart_edma_handle

#include <fsl_uart_edma.h> UART eDMA handle.

Public Members

uart_edma_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.

edma_handle_t *txEdmaHandle
The eDMA TX channel used.

edma_handle_t *rxEdmaHandle
The eDMA RX channel used.

uint8_t nbytes
eDMA minor byte transfer count initially configured.

volatile uint8_t txState
TX transfer state.

volatile uint8_t rxState
RX transfer state

2.56 VREF: Voltage Reference Driver

status_t VREF_Init(VREF_Type *base, const *vref_config_t* *config)

Enables the clock gate and configures the VREF module according to the configuration structure.

This function must be called before calling all other VREF driver functions, read/write registers, and configurations with user-defined settings. The example below shows how to set up vref_config_t parameters and how to call the VREF_Init function by passing in these parameters. This is an example.

```
vref_config_t vrefConfig;
vrefConfig.bufferMode = kVREF_ModeHighPowerBuffer;
vrefConfig.enableExternalVoltRef = false;
vrefConfig.enableLowRef = false;
VREF_Init(VREF, &vrefConfig);
```

Parameters

- base – VREF peripheral address.
- config – Pointer to the configuration structure.

Return values

- kStatus_Success – run success.
- kStatus_Timeout – timeout occurs.

void VREF_Deinit(VREF_Type *base)

Stops and disables the clock for the VREF module.

This function should be called to shut down the module. This is an example.

```
vref_config_t vrefUserConfig;
VREF_Init(VREF);
VREF_GetDefaultConfig(&vrefUserConfig);
...
VREF_Deinit(VREF);
```

Parameters

- base – VREF peripheral address.

void VREF_GetDefaultConfig(*vref_config_t* *config)

Initializes the VREF configuration structure.

This function initializes the VREF configuration structure to default values. This is an example.

```
vrefConfig->bufferMode = kVREF_ModeHighPowerBuffer;
vrefConfig->enableExternalVoltRef = false;
vrefConfig->enableLowRef = false;
```

Parameters

- config – Pointer to the initialization structure.

status_t VREF_SetTrimVal(VREF_Type *base, uint8_t trimValue)

Sets a TRIM value for the reference voltage.

This function sets a TRIM value for the reference voltage. Note that the TRIM value maximum is 0x3F.

Parameters

- base – VREF peripheral address.
- trimValue – Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

Return values

- kStatus_Success – run success.
- kStatus_Timeout – timeout occurs.

static inline uint8_t VREF_GetTrimVal(VREF_Type *base)

Reads the value of the TRIM meaning output voltage.

This function gets the TRIM value from the TRM register.

Parameters

- base – VREF peripheral address.

Returns

Six-bit value of trim setting.

status_t VREF_SetTrim2V1Val(VREF_Type *base, uint8_t trimValue)

Sets a TRIM value for the reference voltage (2V1).

This function sets a TRIM value for the reference voltage (2V1). Note that the TRIM value maximum is 0x3F.

Parameters

- base – VREF peripheral address.
- trimValue – Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

Return values

- kStatus_Success – run success.
- kStatus_Timeout – timeout occurs.

static inline uint8_t VREF_GetTrim2V1Val(VREF_Type *base)

Reads the value of the TRIM meaning output voltage (2V1).

This function gets the TRIM value from the VREF_TRM4 register.

Parameters

- base – VREF peripheral address.

Returns

Six-bit value of trim setting.

status_t VREF_SetLowReferenceTrimVal(VREF_Type *base, uint8_t trimValue)

Sets the TRIM value for the low voltage reference.

This function sets the TRIM value for low reference voltage. Note the following.

- The TRIM value maximum is 0x05U
- The values 111b and 110b are not valid/allowed.

Parameters

- base – VREF peripheral address.
- trimValue – Value of the trim register to set output low reference voltage (maximum 0x05U (3-bit)).

Return values

- kStatus_Success – run success.
- kStatus_Timeout – timeout occurs.

static inline uint8_t VREF_GetLowReferenceTrimVal(VREF_Type *base)

Reads the value of the TRIM meaning output voltage.

This function gets the TRIM value from the VREFL_TRM register.

Parameters

- base – VREF peripheral address.

Returns

Three-bit value of the trim setting.

FSL_VREF_DRIVER_VERSION

Version 2.1.3.

VREF_INTERNAL_VOLTAGE_STABLE_TIMEOUT

Max loops to wait for VREF internal voltage stable.

This parameter defines how many loops to check completion before return timeout. If defined as 0, driver will wait forever until completion.

enum _vref_buffer_mode

VREF modes.

Values:

enumerator kVREF_ModeBandgapOnly

Bandgap on only, for stabilization and startup

enumerator kVREF_ModeHighPowerBuffer
High-power buffer mode enabled

enumerator kVREF_ModeLowPowerBuffer
Low-power buffer mode enabled

typedef enum *_vref_buffer_mode* vref_buffer_mode_t
VREF modes.

typedef struct *_vref_config* vref_config_t
The description structure for the VREF module.

VREF_SC_MODE_LV

VREF_SC_REGEN

VREF_SC_VREFEN

VREF_SC_ICOMPEN

VREF_SC_REGEN_MASK

VREF_SC_VREFST_MASK

VREF_SC_VREFEN_MASK

VREF_SC_MODE_LV_MASK

VREF_SC_ICOMPEN_MASK

TRM

VREF_TRM_TRIM

VREF_TRM_CHOPEN_MASK

VREF_TRM_TRIM_MASK

VREF_TRM_CHOPEN_SHIFT

VREF_TRM_TRIM_SHIFT

VREF_SC_MODE_LV_SHIFT

VREF_SC_REGEN_SHIFT

VREF_SC_VREFST_SHIFT

VREF_SC_ICOMPEN_SHIFT

struct _vref_config

#include <fsl_vref.h> The description structure for the VREF module.

Public Members

vref_buffer_mode_t bufferMode

Buffer mode selection

bool enableLowRef

Set VREFL (0.4 V) reference buffer enable or disable

bool enableExternalVoltRef

Select external voltage reference or not (internal)

bool enable2V1VoltRef

Enable Internal Voltage Reference (2.1V)

2.57 WDOG: Watchdog Timer Driver

void WDOG_GetDefaultConfig(*wdog_config_t* *config)

Initializes the WDOG configuration structure.

This function initializes the WDOG configuration structure to default values. The default values are as follows.

```
wdogConfig->enableWdog = true;
wdogConfig->clockSource = kWDOG_LpoClockSource;
wdogConfig->prescaler = kWDOG_ClockPrescalerDivide1;
wdogConfig->workMode.enableWait = true;
wdogConfig->workMode.enableStop = false;
wdogConfig->workMode.enableDebug = false;
wdogConfig->enableUpdate = true;
wdogConfig->enableInterrupt = false;
wdogConfig->enableWindowMode = false;
wdogConfig->windowValue = 0;
wdogConfig->timeoutValue = 0xFFFFU;
```

See also:

wdog_config_t

Parameters

- config – Pointer to the WDOG configuration structure.

void WDOG_Init(WDOG_Type *base, const *wdog_config_t* *config)

Initializes the WDOG.

This function initializes the WDOG. When called, the WDOG runs according to the configuration. To reconfigure WDOG without forcing a reset first, enableUpdate must be set to true in the configuration.

This is an example.

```
wdog_config_t config;
WDOG_GetDefaultConfig(&config);
config.timeoutValue = 0x7ffU;
config.enableUpdate = true;
WDOG_Init(wdog_base,&config);
```

Parameters

- base – WDOG peripheral base address
- config – The configuration of WDOG

void WDOG_Deinit(WDOG_Type *base)

Shuts down the WDOG.

This function shuts down the WDOG. Ensure that the WDOG_STCTRLH.ALLOWUPDATE is 1 which indicates that the register update is enabled.

void WDOG_SetTestModeConfig(WDOG_Type *base, *wdog_test_config_t* *config)

Configures the WDOG functional test.

This function is used to configure the WDOG functional test. When called, the WDOG goes into test mode and runs according to the configuration. Ensure that the WDOG_STCTRLH.ALLOWUPDATE is 1 which means that the register update is enabled.

This is an example.

```
wdog_test_config_t test_config;
test_config.testMode = kWDOG_QuickTest;
test_config.timeoutValue = 0xffffu;
WDOG_SetTestModeConfig(wdog_base, &test_config);
```

Parameters

- base – WDOG peripheral base address
- config – The functional test configuration of WDOG

static inline void WDOG_Enable(WDOG_Type *base)

Enables the WDOG module.

This function write value into WDOG_STCTRLH register to enable the WDOG, it is a write-once register, make sure that the WCT window is still open and this register has not been written in this WCT while this function is called.

Parameters

- base – WDOG peripheral base address

static inline void WDOG_Disable(WDOG_Type *base)

Disables the WDOG module.

This function writes a value into the WDOG_STCTRLH register to disable the WDOG. It is a write-once register. Ensure that the WCT window is still open and that register has not been written to in this WCT while the function is called.

Parameters

- base – WDOG peripheral base address

static inline void WDOG_EnableInterrupts(WDOG_Type *base, uint32_t mask)

Enables the WDOG interrupt.

This function writes a value into the WDOG_STCTRLH register to enable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

- base – WDOG peripheral base address
- mask – The interrupts to enable The parameter can be combination of the following source if defined.
  - kWDOG_InterruptEnable

static inline void WDOG_DisableInterrupts(WDOG_Type *base, uint32_t mask)

Disables the WDOG interrupt.

This function writes a value into the WDOG_STCTRLH register to disable the WDOG interrupt. It is a write-once register. Ensure that the WCT window is still open and the register has not been written to in this WCT while the function is called.

Parameters

- base – WDOG peripheral base address
- mask – The interrupts to disable The parameter can be combination of the following source if defined.
  - kWDOG_InterruptEnable

uint32_t WDOG_GetStatusFlags(WDOG_Type *base)

Gets the WDOG all status flags.

This function gets all status flags.

This is an example for getting the Running Flag.

```
uint32_t status;
status = WDOG_GetStatusFlags (wdog_base) & kWDOG_RunningFlag;
```

See also:

_wdog_status_flags_t

- true: a related status flag has been set.
- false: a related status flag is not set.

Parameters

- base – WDOG peripheral base address

Returns

State of the status flag: asserted (true) or not-asserted (false).

void WDOG_ClearStatusFlags(WDOG_Type *base, uint32_t mask)

Clears the WDOG flag.

This function clears the WDOG status flag.

This is an example for clearing the timeout (interrupt) flag.

```
WDOG_ClearStatusFlags(wdog_base,kWDOG_TimeoutFlag);
```

Parameters

- base – WDOG peripheral base address
- mask – The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag

static inline void WDOG_SetTimeoutValue(WDOG_Type *base, uint32_t timeoutCount)

Sets the WDOG timeout value.

This function sets the timeout value. It should be ensured that the time-out value for the WDOG is always greater than 2xWCT time + 20 bus clock cycles. This function writes a value into WDOG_TOVALH and WDOG_TOVALL registers which are wirte-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

Parameters

- base – WDOG peripheral base address
- timeoutCount – WDOG timeout value; count of WDOG clock tick.

static inline void WDOG_SetWindowValue(WDOG_Type *base, uint32_t windowValue)

Sets the WDOG window value.

This function sets the WDOG window value. This function writes a value into WDOG_WINH and WDOG_WINL registers which are wirte-once. Ensure the WCT window is still open and the two registers have not been written to in this WCT while the function is called.

Parameters

- base – WDOG peripheral base address
- windowValue – WDOG window value.

static inline void WDOG_Unlock(WDOG_Type *base)

Unlocks the WDOG register written.

This function unlocks the WDOG register written. Before starting the unlock sequence and following configuration, disable the global interrupts. Otherwise, an interrupt may invalidate the unlocking sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

- base – WDOG peripheral base address

void WDOG_Refresh(WDOG_Type *base)

Refreshes the WDOG timer.

This function feeds the WDOG. This function should be called before the WDOG timer is in timeout. Otherwise, a reset is asserted.

Parameters

- base – WDOG peripheral base address

static inline uint16_t WDOG_GetResetCount(WDOG_Type *base)

Gets the WDOG reset count.

This function gets the WDOG reset count value.

Parameters

- base – WDOG peripheral base address

Returns
WDOG reset count value.

static inline void WDOG_ClearResetCount(WDOG_Type *base)
Clears the WDOG reset count.

This function clears the WDOG reset count value.

Parameters

- base – WDOG peripheral base address

FSL_WDOG_DRIVER_VERSION
Defines WDOG driver version 2.0.3.

WDOG_FIRST_WORD_OF_UNLOCK
First word of unlock sequence

WDOG_SECOND_WORD_OF_UNLOCK
Second word of unlock sequence

WDOG_FIRST_WORD_OF_REFRESH
First word of refresh sequence

WDOG_SECOND_WORD_OF_REFRESH
Second word of refresh sequence

enum _wdog_clock_source
Describes WDOG clock source.

Values:

enumerator kWDOG_LpoClockSource
WDOG clock sourced from LPO

enumerator kWDOG_AlternateClockSource
WDOG clock sourced from alternate clock source

enum _wdog_clock_prescaler
Describes the selection of the clock prescaler.

Values:

enumerator kWDOG_ClockPrescalerDivide1
Divided by 1

enumerator kWDOG_ClockPrescalerDivide2
Divided by 2

enumerator kWDOG_ClockPrescalerDivide3
Divided by 3

enumerator kWDOG_ClockPrescalerDivide4
Divided by 4

enumerator kWDOG_ClockPrescalerDivide5
Divided by 5

enumerator kWDOG_ClockPrescalerDivide6
Divided by 6

enumerator kWDOG_ClockPrescalerDivide7
Divided by 7

enumerator kWDOG_ClockPrescalerDivide8
Divided by 8

enum _wdog_test_mode
Describes WDOG test mode.
Values:

enumerator kWDOG_QuickTest
Selects quick test

enumerator kWDOG_ByteTest
Selects byte test

enum _wdog_tested_byte
Describes WDOG tested byte selection in byte test mode.
Values:

enumerator kWDOG_TestByte0
Byte 0 selected in byte test mode

enumerator kWDOG_TestByte1
Byte 1 selected in byte test mode

enumerator kWDOG_TestByte2
Byte 2 selected in byte test mode

enumerator kWDOG_TestByte3
Byte 3 selected in byte test mode

enum _wdog_interrupt_enable_t
WDOG interrupt configuration structure, default settings all disabled.
This structure contains the settings for all of the WDOG interrupt configurations.
Values:

enumerator kWDOG_InterruptEnable
WDOG timeout generates an interrupt before reset

enum _wdog_status_flags_t

WDOG status flags.

This structure contains the WDOG status flags for use in the WDOG functions.

Values:

enumerator kWDOG_RunningFlag

Running flag, set when WDOG is enabled

enumerator kWDOG_TimeoutFlag

Interrupt flag, set when an exception occurs

typedef enum *_wdog_clock_source* wdog_clock_source_t

Describes WDOG clock source.

typedef struct *_wdog_work_mode* wdog_work_mode_t

Defines WDOG work mode.

typedef enum *_wdog_clock_prescaler* wdog_clock_prescaler_t

Describes the selection of the clock prescaler.

typedef struct *_wdog_config* wdog_config_t

Describes WDOG configuration structure.

typedef enum *_wdog_test_mode* wdog_test_mode_t

Describes WDOG test mode.

typedef enum *_wdog_tested_byte* wdog_tested_byte_t

Describes WDOG tested byte selection in byte test mode.

typedef struct *_wdog_test_config* wdog_test_config_t

Describes WDOG test mode configuration structure.

WDOG_WCT_INSTRUCITON_COUNT

< Watchdog configuration time window

struct _wdog_work_mode

#include <fsl_wdog.h> Defines WDOG work mode.

Public Members

bool enableWait

Enables or disables WDOG in wait mode

bool enableStop

Enables or disables WDOG in stop mode

bool enableDebug
Enables or disables WDOG in debug mode

struct _wdog_config
#include <fsl_wdog.h> Describes WDOG configuration structure.

Public Members

bool enableWdog
Enables or disables WDOG

wdog_clock_source_t clockSource
Clock source select

wdog_clock_prescaler_t prescaler
Clock prescaler value

wdog_work_mode_t workMode
Configures WDOG work mode in debug stop and wait mode

bool enableUpdate
Update write-once register enable

bool enableInterrupt
Enables or disables WDOG interrupt

bool enableWindowMode
Enables or disables WDOG window mode

uint32_t windowValue
Window value

uint32_t timeoutValue
Timeout value

struct _wdog_test_config
#include <fsl_wdog.h> Describes WDOG test mode configuration structure.

Public Members

wdog_test_mode_t testMode
Selects test mode

wdog_tested_byte_t testedByte
Selects tested byte in byte test mode

uint32_t timeoutValue
Timeout value

Chapter 3

Middleware

3.1 File System

3.1.1 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.

3.2 Motor Control

3.2.1 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.

3.3 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.

3.3.1 Quick links

- *Release Notes*

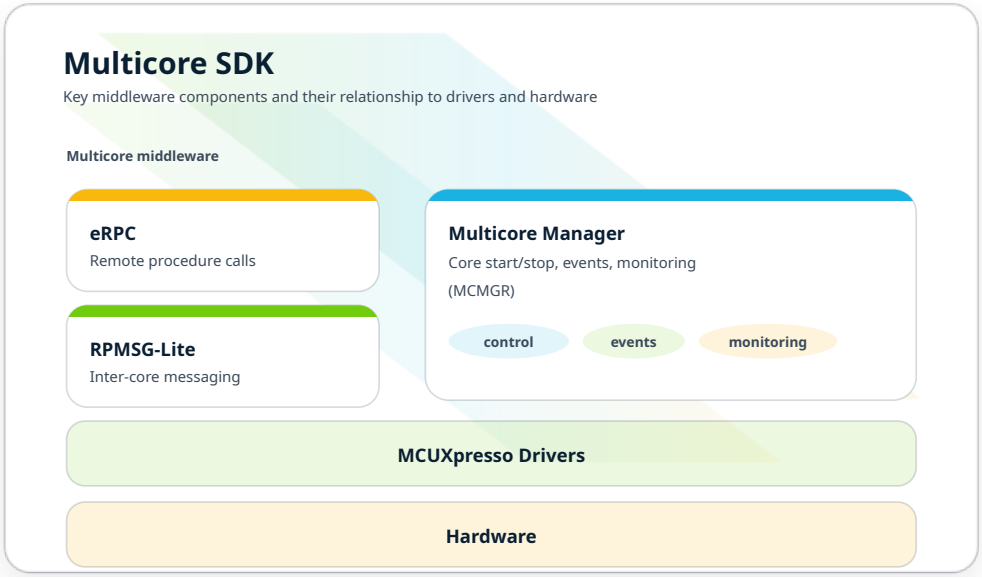


Fig. 1: Multicore SDK software stack (simplified).

- *Getting Started*
- *Multicore SDK Changelog*

3.3.2 Components

Component	Description
RPMSG-Lite	Lightweight implementation of the Remote Processor Messaging (RPMsg) protocol for inter-core messaging.
Multicore Manager	Services for multicore systems (core start/stop, events, and monitoring).
eRPC	Embedded RPC system suitable for multicore and multiprocessor designs.

MCSDK Release Notes

Multicore SDK (MCSDK) Release Notes Overview

These are the release notes for the NXP **Multicore Software Development Kit (MCSDK)**.

Item	Value
MCSDK version	**26.09.00**
Scope	Multicore middleware components and examples
Location in MCUXpresso SDK	<MCUXpressoSDK_install_dir>/middleware/multicore/

The MCSDK is combined with the MCUXpresso SDK to provide a framework for developing multicore applications on NXP devices.

What's new

Component updates

Review detailed changes per component:

- eRPC: *CHANGELOG*
- RPMSG-Lite: *CHANGELOG*
- Multicore Manager (MCMGR): *CHANGELOG*

Supported evaluation boards

Boards with enabled multicore example coverage in this release:

Multicore examples
LPCXpresso55S69FRDM-K32L3A6MIMXRT1170-EVKBMIMXRT1160-EVKMIMXRT1180-EVKMCX-N5XX-EVKMCX-N9XX-EVKFRDM-MCXN947MIMXRT700-EVKKW47-EVKKW47-LOCFRDM-MCXW72MCX-W72-EVKFRDM-IMXRT1186

Development tools

The MCSDK is compiled and tested with the toolchains described in the MCUXpresso SDK documentation:

- https://mcuxpresso.nxp.com/mcuxsdk/latest/html/gsd/installation.html#toolchain

Release contents

This table describes the release contents. Not all MCUXpresso SDK packages contain the full set of these components.

Deliverable	Location
Multicore SDK location (<MCSDK_dir>)	<MCUXpressoSDK_install_dir>/middleware/multicore/
Documentation	<MCSDK_dir>/mcuxsdk-doc/
Embedded Remote Procedure Call (eRPC)	<MCSDK_dir>/erpc/
Multicore Manager (MCMGR)	<MCSDK_dir>/mcmgr/
RPMSG-Lite	<MCSDK_dir>/rpmsg-lite/
Multicore demo applications	<MCUXpressoSDK_install_dir>/examples/multicore_examples/

MCSDK component overview

Together, the MCSDK and the MCUXpresso SDK form a framework for developing software for NXP multicore devices. The MCSDK release includes:

- Embedded Remote Procedure Call (eRPC)
- Multicore Manager (MCMGR)
- Remote Processor Messaging – Lite (RPMSG-Lite)

The MCSDK also includes documentation and demo applications for multicore communication.

Demo applications

The demo applications demonstrate MCSDK component usage on supported development boards.

Examples are located under:

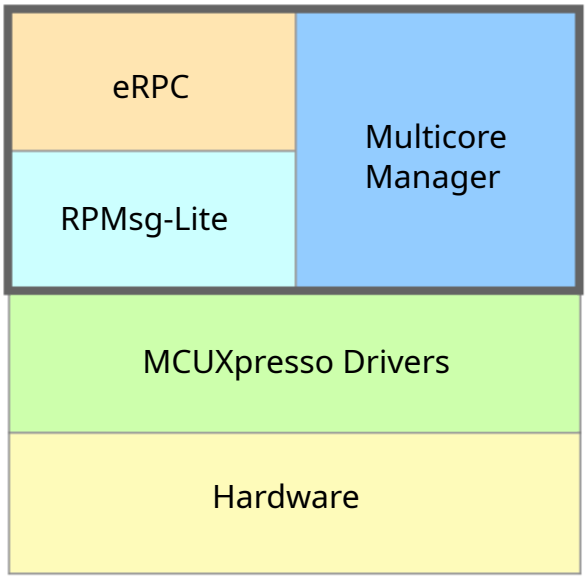
- Multicore: <MCUXpressoSDK_install_dir>/examples/multicore_examples/

Multicore examples
freertos_message_buffershello_worldmulticore_managerrpmsg_lite_pingpongrpmsg_lite_pingpong_rtosrpmsg_lite_

MCSDK Getting Started

Getting Started with Multicore SDK (MCSDK) Overview

The **Multicore Software Development Kit (MCSDK)** provides software components for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to provide a framework for developing multicore applications.



All MCSDK-related files are located in:

- <MCUXpressoSDK_install_dir>/middleware/multicore/

For supported toolchain versions, see the release notes.

Quick links

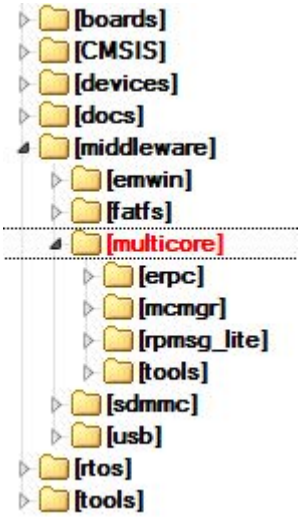
- MCSDK Release Notes
- *MCSDK Changelog*
- Component repositories (within MCSDK):
  - RPMSG-Lite
  - Multicore Manager (MCMGR)
  - eRPC
- eRPC detailed Getting Started guide:

– eRPC Getting Started

MCSDK components

The MCSDK consists of the following components:

- **Embedded Remote Procedure Call (eRPC):** Library + code generator implementing transparent function calls to services running on another core.
- **Multicore Manager (MCMGR):** Core management services (start/stop secondary cores, events, monitoring).
- **Remote Processor Messaging – Lite (RPMSG-Lite):** Lightweight inter-processor messaging library based on RPMsg.



Where things live

Item	Path
MCSDK root	<MCUXpressoSDK_install_dir>/middleware/multicore/
eRPC	<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/
MCMGR	<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/
RPMSG-Lite	<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg-lite/
MCSDK documentation	<MCUXpressoSDK_install_dir>/middleware/multicore/mcuxsdk-doc/
Multicore examples	<MCUXpressoSDK_install_dir>/examples/multicore_examples/

MCSDK demo applications

Multicore example applications are stored together with other MCUXpresso SDK examples.

Location	Folder
Multicore example projects	<MCUXpressoSDK_install_dir>/examples/multicore_examples/<application_name>/

Each example application contains a README describing setup, required images, and runtime behavior.

Inter-Processor Communication (IPC) levels

The MCSDK provides several mechanisms for Inter-Processor Communication (IPC). Pick the lightest one that matches your requirements.

IPC using low-level drivers

NXP multicore SoCs commonly provide dedicated IPC peripherals (Mailbox for some LPC parts, Messaging Unit (MU) for some Kinetis and i.MX parts).

This is the most lightweight approach: using the Mailbox/MU driver API functions, it is possible to pass a value from core to core via registers (scalar or pointer to shared memory) and trigger inter-core interrupts for notifications.

For details about individual driver API functions, see the MCUXpresso SDK API Reference Manual of the specific device.

Messaging mechanism (RPMSG-Lite)

On top of Mailbox/MU drivers, a messaging system can be implemented, allowing messages to be sent between endpoints created on each CPU.

RPMSG-Lite provides this capability and is the preferred MCUXpresso SDK messaging library. It implements ring buffers in shared memory for message exchange.

This costs additional code and data memory, but provides a unified messaging API and helps portability across devices/RTOSes.

Remote procedure calls (eRPC)

To facilitate IPC further, eRPC allows remote function invocation via a simple local function call.

Using different transport layers, it can communicate:

- between individual cores of multicore SoCs (commonly via RPMSG-Lite)
- between separate processors (SPI, UART, TCP/IP)

If the communicating peer is a Linux user-space application, the generated code can be in C/C++ or Python.

The IPC levels demonstrate the scalability of the MCSDK. Based on application needs, different IPC techniques can be used depending on complexity, speed, memory, and system design.

Changelog Multicore SDK

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

[26.09.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.3.0
  - RPMsg-Lite v5.5.0

[26.06.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0

  - Multicore Manager (MCMgr) v5.2.0
  - RPMsg-Lite v5.4.1

[26.03.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.1.0
  - RPMsg-Lite v5.4.0

[25.12.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.2
  - RPMsg-Lite v5.3.0

[25.09.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.1
  - RPMsg-Lite v5.2.1

[25.06.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.0
  - RPMsg-Lite v5.2.0

[25.03.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.7
  - RPMsg-Lite v5.1.4

[24.12.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.6
  - RPMsg-Lite v5.1.3

[2.16.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.13.0
  - eRPC generator (erpcgen) v1.13.0
  - Multicore Manager (MCMgr) v4.1.5
  - RPMsg-Lite v5.1.2

[2.15.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.12.0
  - eRPC generator (erpcgen) v1.12.0
  - Multicore Manager (MCMgr) v4.1.5
  - RPMsg-Lite v5.1.1

[2.14.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.11.0
  - eRPC generator (erpcgen) v1.11.0
  - Multicore Manager (MCMgr) v4.1.4
  - RPMsg-Lite v5.1.0

[2.13.0_imxrt1180a0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.10.0
  - eRPC generator (erpcgen) v1.10.0
  - Multicore Manager (MCMgr) v4.1.3
  - RPMsg-Lite v5.0.0

[2.13.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.10.0
  - eRPC generator (erpcgen) v1.10.0
  - Multicore Manager (MCMgr) v4.1.3
  - RPMsg-Lite v5.0.0

[2.12.0_imx93]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.1
  - eRPC generator (erpcgen) v1.9.1
  - Multicore Manager (MCMgr) v4.1.2
  - RPMsg-Lite v4.0.1

[2.12.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.1
  - eRPC generator (erpcgen) v1.9.1
  - Multicore Manager (MCMgr) v4.1.2
  - RPMsg-Lite v4.0.0

[2.11.1]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.0
  - eRPC generator (erpcgen) v1.9.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.2.1

[2.11.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.9.0
  - eRPC generator (erpcgen) v1.9.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.2.0

[2.10.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.8.1
  - eRPC generator (erpcgen) v1.8.1
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.1.2

[2.9.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.8.0
  - eRPC generator (erpcgen) v1.8.0
  - Multicore Manager (MCMgr) v4.1.1
  - RPMsg-Lite v3.1.1

[2.8.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.4
  - eRPC generator (erpcgen) v1.7.4
  - Multicore Manager (MCMgr) v4.1.0
  - RPMsg-Lite v3.1.0

[2.7.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.3
  - eRPC generator (erpcgen) v1.7.3
  - Multicore Manager (MCMgr) v4.1.0
  - RPMsg-Lite v3.0.0

[2.6.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.2
  - eRPC generator (erpcgen) v1.7.2
  - Multicore Manager (MCMgr) v4.0.3
  - RPMsg-Lite v2.2.0

[2.5.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.1
  - eRPC generator (erpcgen) v1.7.1
  - Multicore Manager (MCMgr) v4.0.2
  - RPMsg-Lite v2.0.2

[2.4.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.7.0
  - eRPC generator (erpcgen) v1.7.0
  - Multicore Manager (MCMgr) v4.0.1
  - RPMsg-Lite v2.0.1

[2.3.1]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.6.0
  - eRPC generator (erpcgen) v1.6.0
  - Multicore Manager (MCMgr) v4.0.0
  - RPMsg-Lite v1.2.0

[2.3.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.5.0
  - eRPC generator (erpcgen) v1.5.0
  - Multicore Manager (MCMgr) v3.0.0
  - RPMsg-Lite v1.2.0

[2.2.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.4.0
  - eRPC generator (erpcgen) v1.4.0
  - Multicore Manager (MCMgr) v2.0.1
  - RPMsg-Lite v1.1.0

[2.1.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.3.0
  - eRPC generator (erpcgen) v1.3.0

[2.0.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.2.0
  - eRPC generator (erpcgen) v1.2.0
  - Multicore Manager (MCMgr) v2.0.0
  - RPMsg-Lite v1.0.0

[1.1.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.1.0
  - Multicore Manager (MCMgr) v1.1.0
  - Open-AMP / RPMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev01

[1.0.0]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.0.0
  - Multicore Manager (MCMgr) v1.0.0
  - Open-AMP / RPMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev00

RPMSG-Lite

Quick links

- *Changelog*
- API documentation (Doxygen): https://nxp-mcuxpresso.github.io/rpmsg-lite/

This documentation describes the RPMsg-Lite component, which is a lightweight implementation of the Remote Processor Messaging (RPMsg) protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system.

Compared to the RPMsg implementation of the Open Asymmetric Multi Processing (OpenAMP) framework (https://github.com/OpenAMP/open-amp), the RPMsg-Lite offers a code size reduction, API simplification, and improved modularity. On smaller Cortex-M0+ based systems, it is recommended to use RPMsg-Lite.

The RPMsg-Lite is an open-source component developed by NXP Semiconductors and released under the BSD-compatible license.

For overview please read RPMSG-Lite VirtIO Overview.

For RPMSG-Lite Design Considerations please read RPMSG-Lite Design Considerations.

Motivation to create RPMsg-Lite

There are multiple reasons why RPMsg-Lite was developed. One reason is the need for the small footprint of the RPMsg protocol-compatible communication component, another reason is the simplification of extensive API of OpenAMP RPMsg implementation.

RPMsg protocol was not documented, and its only definition was given by the Linux Kernel and legacy OpenAMP implementations. This has changed with [1] which is a standardization protocol allowing multiple different implementations to coexist and still be mutually compatible.

Small MCU-based systems often do not implement dynamic memory allocation. The creation of static API in RPMsg-Lite enables another reduction of resource usage. Not only does the dynamic allocation adds another 5 KB of code size, but also communication is slower and less deterministic, which is a property introduced by dynamic memory. The following table shows some rough comparison data between the OpenAMP RPMsg implementation and new RPMsg-Lite implementation:

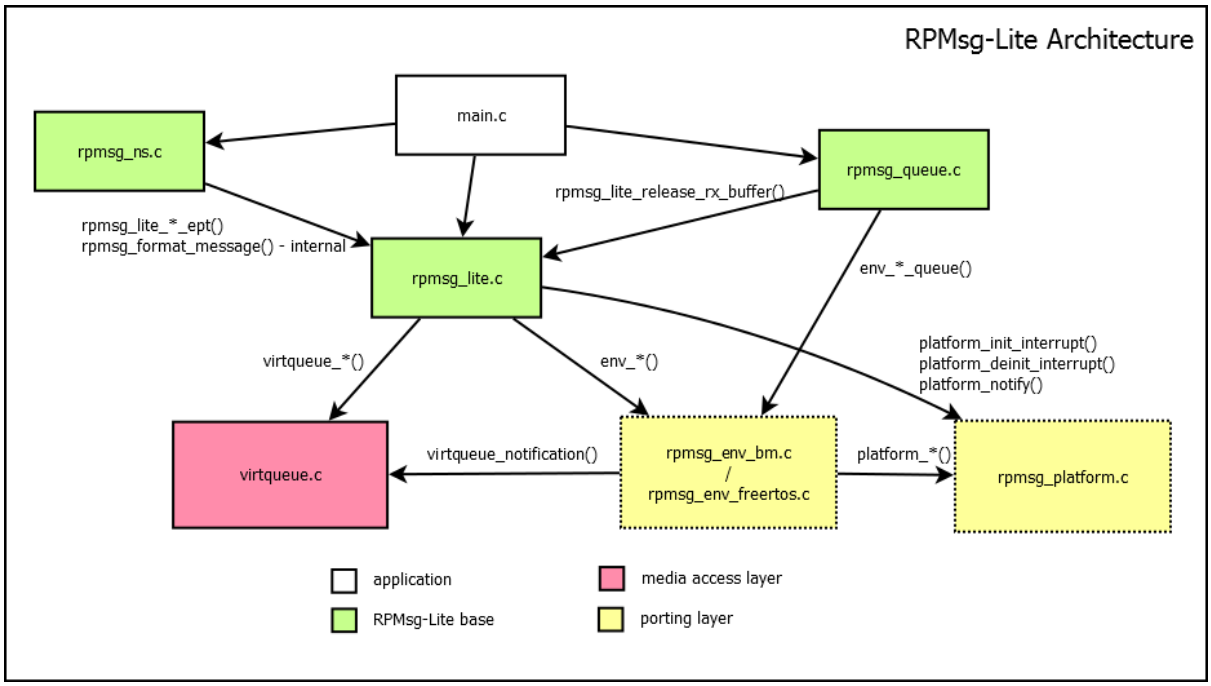
Component / Configuration	Flash [B]	RAM [B]
OpenAMP RPMsg / Release (reference)	5547	456 + dynamic
RPMsg-Lite / Dynamic API, Release	3462	56 + dynamic
Relative Difference [%]	~62.4%	~12.3%
RPMsg-Lite / Static API (no malloc), Release	2926	352
Relative Difference [%]	~52.7%	~77.2%

Implementation

The implementation of RPMsg-Lite can be divided into three sub-components, from which two are optional. The core component is situated in rpmsg_lite.c. Two optional components are used to implement a blocking receive API (in rpmsg_queue.c) and dynamic “named” endpoint creation and deletion announcement service (in rpmsg_ns.c).

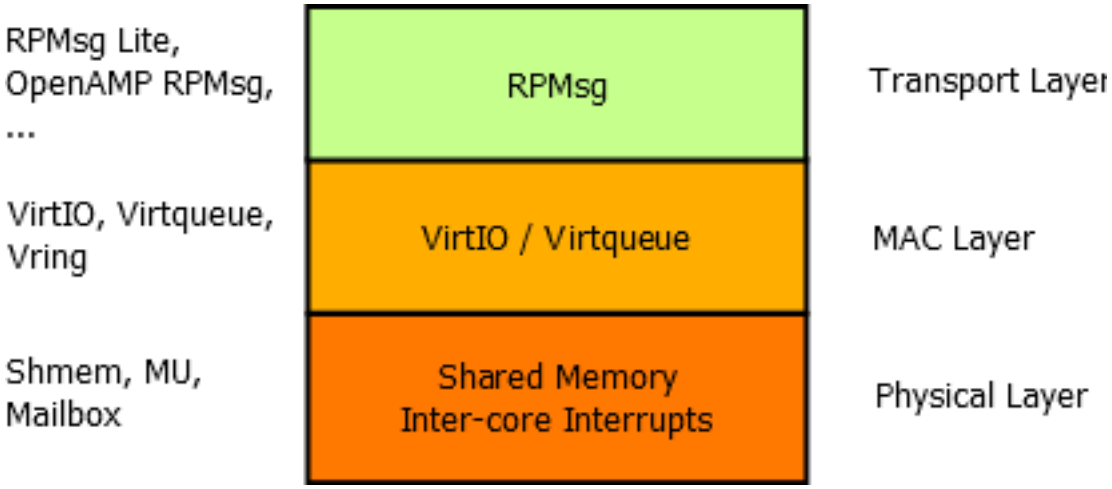
The actual “media access” layer is implemented in virtqueue.c, which is one of the few files shared with the OpenAMP implementation. This layer mainly defines the shared memory model, and internally defines used components such as vring or virtqueue.

The porting layer is split into two sub-layers: the environment layer and the platform layer. The first sublayer is to be implemented separately for each environment. (The bare metal environment already exists and is implemented in rpmsg_env_bm.c, and the FreeRTOS environment is implemented in rpmsg_env_freertos.c etc.) Only the source file, which matches the used environment, is included in the target application project. The second sublayer is implemented in rpmsg_platform.c and defines low-level functions for interrupt enabling, disabling, and triggering mainly. The situation is described in the following figure:



RPMsg-Lite core sub-component

This subcomponent implements a blocking send API and callback-based receive API. The RPMsg protocol is part of the transport layer. This is realized by using so-called endpoints. Each endpoint can be assigned a different receive callback function. However, it is important to notice that the callback is executed in an interrupt environment in current design. Therefore, certain actions like memory allocation are discouraged to execute in the callback. The following figure shows the role of RPMsg in an ISO/OSI-like layered model:



Queue sub-component (optional)

This subcomponent is optional and requires implementation of the env_*_queue() functions in the environment porting layer. It uses a blocking receive API, which is common in RTOS-environments. It supports both copy and nocopy blocking receive functions.

Name Service sub-component (optional)

This subcomponent is a minimum implementation of the name service which is present in the Linux Kernel implementation of RPMsg. It allows the communicating node both to send announcements about “named” endpoint (in other words, channel) creation or deletion and to receive these announcement taking any user-defined action in an application callback. The endpoint address used to receive name service announcements is arbitrarily fixed to be 53 (0x35).

Usage

The application should put the /rpmsg_lite/lib/include directory to the include path and in the application, include either the rpmsg_lite.h header file, or optionally also include the rpmsg_queue.h and/or rpmsg_ns.h files. Both porting sublayers should be provided for you by NXP, but if you plan to use your own RTOS, all you need to do is to implement your own environment layer (in other words, rpmsg_env_myrtos.c) and to include it in the project build.

The initialization of the stack is done by calling the rpmsg_lite_master_init() on the master side and the rpmsg_lite_remote_init() on the remote side. This initialization function must be called prior to any RPMsg-Lite API call. After the init, it is wise to create a communication endpoint, otherwise communication is not possible. This can be done by calling the rpmsg_lite_create_ept() function. It optionally accepts a last argument, where an internal context of the endpoint is created, just in case the RL_USE_STATIC_API option is set to 1. If not, the stack internally calls env_alloc() to allocate dynamic memory for it. In case a callback-based receiving is to be used, an ISR-callback is registered to each new endpoint with user-defined callback data pointer. If a blocking receive is desired (in case of RTOS environment), the rpmsg_queue_create() function must be called before calling rpmsg_lite_create_ept(). The queue handle is passed to the endpoint creation function as a callback data argument and the callback function is set to rpmsg_queue_rx_cb(). Then, it is possible to use rpmsg_queue_receive() function to listen on a queue object for incoming messages. The rpmsg_lite_send() function is used to send messages to the other side.

The RPMsg-Lite also implements no-copy mechanisms for both sending and receiving operations. These methods require specifics that have to be considered when used in an application.

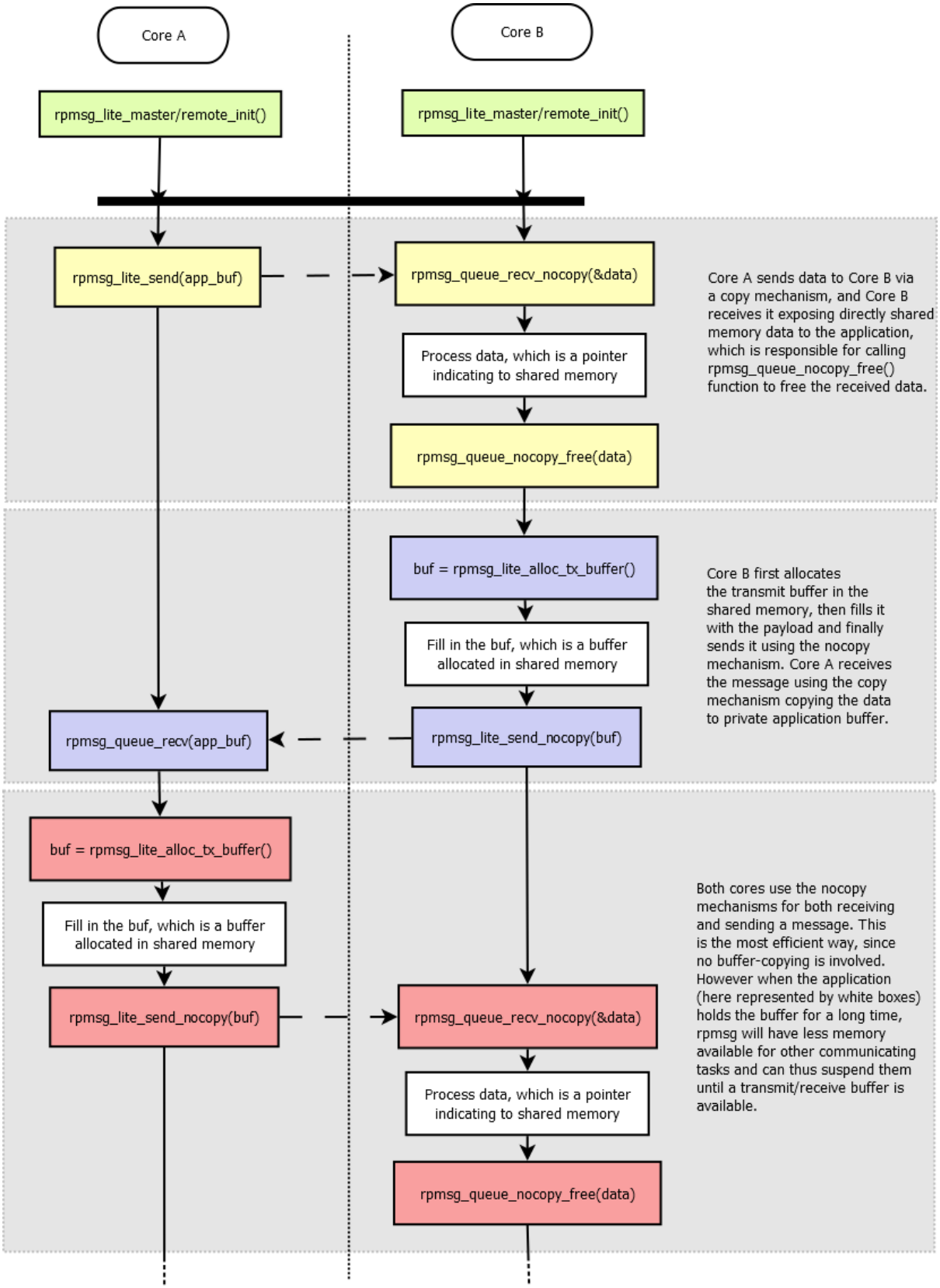
no-copy-send mechanism: This mechanism allows sending messages without the cost for copying data from the application buffer to the RPMsg/virtio buffer in the shared memory. The sequence of no-copy sending steps to be performed is as follows:

- Call the rpmsg_lite_alloc_tx_buffer() function to get the virtio buffer and provide the buffer pointer to the application.
- Fill the data to be sent into the pre-allocated virtio buffer. Ensure that the filled data does not exceed the buffer size (provided as the rpmsg_lite_alloc_tx_buffer() size output parameter).
- Call the rpmsg_lite_send_nocopy() function to send the message to the destination endpoint. Consider the cache functionality and the virtio buffer alignment. See the rpmsg_lite_send_nocopy() function description below.

no-copy-receive mechanism: This mechanism allows reading messages without the cost for copying data from the virtio buffer in the shared memory to the application buffer. The sequence of no-copy receiving steps to be performed is as follows:

- Call the rpmsg_queue_recv_nocopy() function to get the virtio buffer pointer to the received data.
- Read received data directly from the shared memory.
- Call the rpmsg_queue_nocopy_free() function to release the virtio buffer and to make it available for the next data transfer.

The user is responsible for destroying any RPMsg-Lite objects he has created in case of deinitialization. In order to do this, the function rpmsg_queue_destroy() is used to destroy a queue, rpmsg_lite_destroy_ept() is used to destroy an endpoint and finally, rpmsg_lite_deinit() is used to deinitialize the RPMsg-Lite intercore communication stack. Deinitialize all endpoints using a queue before deinitializing the queue. Otherwise, you are actively invalidating the used queue handle, which is not allowed. RPMsg-Lite does not check this internally, since its main aim is to be lightweight.



Examples

RPMsg_Lite multicore examples are part of NXP MCUXpressoSDK packages. Visit https://mcuxpresso.nxp.com to configure, build and download these packages. To get the board list with multicore support (RPMsg_Lite included) use filtering based on Middleware and search for

'multicore' string. Once the selected package with the multicore middleware is downloaded, see

<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples for RPMsg_Lite multicore examples with 'rpmsg_lite_' name prefix.

Another way of getting NXP MCUXpressoSDK RPMsg_Lite multicore examples is using the mcuxsdk-manifests Github repo. Follow the description how to use the West tool to clone and update the mcuxsdk-manifests repo in readme section. Once done the armgcc rpmsg_lite examples can be found in

mcuxsdk/examples/_<board_name>/multicore_examples

You can use the evkmimxrt1170 as the board_name for instance. Similar to MCUXpressoSDK packages the RPMsg_Lite examples use the 'rpmsg_lite_' name prefix.

Notes

Environment layers implementation

Several environment layers are provided in lib/rpmsg_lite/porting/environment folder. Not all of them are fully tested however. Here is the list of environment layers that passed testing:

- rpmsg_env_bm.c
- rpmsg_env_freertos.c
- rpmsg_env_xos.c
- rpmsg_env_threadx.c

The rest of environment layers has been created and used in some experimental projects, it has been running well at the time of creation but due to the lack of unit testing there is no guarantee it is still fully functional.

Shared memory configuration

It is important to correctly initialize/configure the shared memory for data exchange in the application. The shared memory must be accessible from both the master and the remote core and it needs to be configured as Non-Cacheable memory. Dedicated shared memory section in liker file is also a good practise, it is recommended to use linker files from MCUXpressSDK packages for NXP devices based applications. It needs to be ensured no other application part/component is unintentionally accessing this part of memory.

Configuration options

The RPMsg-Lite can be configured at the compile time. The default configuration is defined in the rpmsg_default_config.h header file. This configuration can be customized by the user by including rpmsg_config.h file with custom settings. The following table summarizes all possible RPMsg-Lite configuration options.

Configuration option	Default value	Usage
RL_MS_PE	(1)	Delay in milliseconds used in non-blocking API functions for polling.
RL_BUFFE	(496)	Size of the buffer payload, it must be more than 1 byte, and has to be word align (including rpmsg header size 16 bytes), if not it will be aligned up
RL_BUFFE	(2)	Number of the buffers, it must be power of two (2, 4, ...)
RL_API_H	(1)	Zero-copy API functions enabled/disabled.
RL_USE_S	(0)	Static API functions (no dynamic allocation) enabled/disabled.
RL_USE_D	(0)	Memory cache management of shared memory. Use in case of data cache is enabled for shared memory.
RL_CLEAF	(0)	Clearing used buffers before returning back to the pool of free buffers enabled/disabled.
RL_USE_M	(0)	When enabled IPC interrupts are managed by the Multicore Manager (IPC interrupts router), when disabled RPMsg-Lite manages IPC interrupts by itself.
RL_USE_E	(0)	When enabled the environment layer uses its own context. Required for some environments (QNX). The default value is 0 (no context, saves some RAM).
RL_DEBU(	(0)	When enabled buffer pointers passed to rpmsg_lite_send_nocopy() and rpmsg_lite_release_rx_buffer() functions (enabled by RL_API_HAS_ZEROCOPY config) are checked to avoid passing invalid buffer pointer. The default value is 0 (disabled). Do not use in RPMsg-Lite to Linux configuration.
RL_ALLO\	(0)	When enabled the opposite side is notified each time received buffers are consumed and put into the queue of available buffers. Enable this option in RPMsg-Lite to Linux configuration to allow unblocking of the Linux blocking send. The default value is 0 (RPMsg-Lite to RPMsg-Lite communication).
RL_ALLO\	(0)	It allows to define custom shared memory configuration and replacing the shared memory related global settings from rpmsg_config.h This is useful when multiple instances are running in parallel but different shared memory arrangement (vring size & alignment, buffers size & count) is required. The default value is 0 (all RPMsg_Lite instances use the same shared memory arrangement as defined by common config macros).
RL_ASSER	see rpmsg	Assert implementation.

How to format rpmsg-lite code

To format code, use the application developed by Google, named *clang-format*. This tool is part of the llvm project. Currently, the clang-format 10.0.0 version is used for rpmsg-lite. The set of style settings used for clang-format is defined in the `.clang-format` file, placed in a root of the rpmsg-lite directory where Python script `run_clang_format.py` can be executed. This script executes the application named *clang-format.exe*. You need to have the path of this application in the OS's environment path, or you need to change the script.

References

[1] M. Novak, M. Cingel, Lockless Shared Memory Based Multicore Communication Protocol

Copyright © 2016 Freescale Semiconductor, Inc. Copyright © 2016-2025 NXP

Changelog RPMSG-Lite

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

v5.5.0

Added

- Added MCXE32B dual Cortex-M7 parts support.
- Added rpmsg_lite_are_all_buffers_consumed() API to report when all locally allocated TX buffers were consumed by the peer and returned to the local free buffer pool, with test coverage in tests/03_send_receive for both sides.
- Added i.MX937 parts support
- Removed obsolete unused k32w1 platform layer

v5.4.1

Fixed

- Corrected isr_counter3 handling in RT700 porting layer platform_deinit_interrupt implementation.

v5.4.0

Added

- Added i.MX93W parts support
- Added i.MX952 parts support
- Added mcxl14x parts support
- Added mcxw70 parts support
- Zephyr rpmsg-lite samples add support for RT600 and RT700 DSP cores

Changed

- Port platform_get_custom_shmem_config change return value from uint32_t to int32_t for error handling consistency.

Fixed

- Updated unit tests code to allow correct GCOV code coverage measurement.
- Added missing dsb instructions into all ISRs for cm4/cm7 based platform layers.
- Fixed CERT EXP34-C by adding __attribute__((noreturn)) to RL_HANG(void) function.
- **MISRA C-2012 Rule 14.3 compliance**: Simplified RL_WORD_ALIGN_UP and RL_WORD_ALIGN_DOWN macros to eliminate invariant controlling expression violations (CID 5011148)
  - Replaced ternary operator implementations with standard bitwise alignment idioms
  - Fixed hardcoded 4U constant in RL_WORD_ALIGN_UP to use RL_WORD_SIZE for better portability
- Fixed several MISRA compliance violations

v5.3.0

Added

- RT700 porting layer added support to send rpmsg messages between CM33_0 <-> Hifi1 and CM33_1 <-> Hifi4 cores.
- Add new platform macro RL_PLATFORM_MAX_ISR_COUNT this will set number of IRQ count per platform. This macro is then used in environment layers to set isr_table size where irq handles are registered. It size should match the bit length of VQ_ID so all combinations can fit into table.
- Unit tests updated to improve code coverage, new unit tests added covering static allocations in rtos environment layers.

Fixed

- virtio.h removed typedef uint8_t boolean and in its place use standard C99 bool type to avoid potential type conflicts.
- env_acquire_sync_lock() and env_release_sync_lock() synchronization primitives removed
- Kconfig consolidation, when RL_ALLOW_CUSTOM_SHMEM_CONFIG enabled the platform_get_custom_shmem_config() function needs to be implemented in platform layer to provide custom shared memory configuration for RPMsg-Lite instance.

v5.2.1

Added

- Doc added RPMSG-Lite VirtIO Overview
- Doc added RPSMG-Lite Design Considerations
- Added frdmimxrt1186 unit testing

Changed

- Remove limitation that RL_BUFFER_SIZE needs to be power of 2. It just has to be more than 16 bytes, e.g. 16 bytes of rpmsg header and payload size at least 1 byte and word aligned, if not it will be aligned up.

Fixed

- Fixed CERT-C INT31-C violation in platform_notify function in rpmsg_platform.c for imxrt700_m33, imxrt700_hifi4, imxrt700_hifi1 platforms

v5.2.0

Added

- Add MCXL20 porting layer and unit testing
- New utility macro RL_CALCULATE_BUFFER_COUNT_DOWN_SAFE to safely determine maximum buffer count within shared memory while preventing integer underflow.
- RT700 platform add support for MCMGR in DSPs

Changed

- Change rpmsg_platform.c to support new MCMGR API
- Improved input validation in initialization functions to properly handle insufficient memory size conditions.
- Refactored repeated buffer count calculation pattern for better code maintainability.

- To make sure that remote has already registered IRQ there is required App level IPC mechanism to notify master about it

Fixed

- Fixed env_wait_for_link_up function to handle timeout in link state checks for baremetal and qnx environment, RL_BLOCK mode can be used to wait indefinitely.
- Fixed CERT-C INT31-C violation by adding compile-time check to ensure RL_PLATFORM_HIGHEST_LINK_ID remains within safe range for 16-bit casting in virtqueue ID creation.
- Fixed CERT-C INT30-C violations by adding protection against unsigned integer underflow in shared memory calculations, specifically in shmem_length - (uint32_t)RL_VRING_OVERHEAD and shmem_length - 2U * shmem_config.vring_size expressions.
- Fixed CERT INT31-C violation in platform_interrupt_disable() and similar functions by replacing unsafe cast from uint32_t to int32_t with a return of 0 constant.
- Fixed unsigned integer underflow in rpmsg_lite_alloc_tx_buffer() where subtracting header size from buffer size could wrap around if buffer was too small, potentially leading to incorrect buffer sizing.
- Fixed CERT-C INT31-C violation in rpmsg_lite.c where size parameter was cast from uint32_t to uint16_t without proper validation.
  - Applied consistent masking approach to both size and flags parameters: (uint16_t)(value & 0xFFFFU).
  - This fix prevents potential data loss when size values exceed 65535.
- Fixed CERT INT31-C violation in env_memset functions by explicitly converting int32_t values to unsigned char using bit masking. This prevents potential data loss or misinterpretation when passing values outside the unsigned char range (0-255) to the standard memset() function.
- Fixed CERT-C INT31-C violations in RPMsg-Lite environment porting: Added validation checks for signed-to-unsigned integer conversions to prevent data loss and misinterpretation.
  - rpmsg_env_freertos.c: Added validation before converting int32_t to UBaseType_t.
  - rpmsg_env_qnx.c: Fixed format string and added validation before assigning to mqstat fields.
  - rpmsg_env_threadx.c: Added validation to prevent integer overflow and negative values.
  - rpmsg_env_xos.c: Added range checking before casting to uint16_t.
  - rpmsg_env_zephyr.c: Added validation before passing values to k_msgq_init.
- Fixed a CERT INT31-C compliance issue in env_get_current_queue_size() function where an unsigned queue count was cast to a signed int32_t without proper validation, which could lead to lost or misinterpreted data if queue size exceeded INT32_MAX.
- Fixed CERT INT31-C violation in rpmsg_platform.c where memcmp() return value (signed int) was compared with unsigned constant without proper type handling.
- Fixed CERT INT31-C violation in rpmsg_platform.c where casting from uint32_t to uint16_t could potentially result in data loss. Changed length variable type from uint16_t to uint32_t to properly handle memory address differences without truncation.
- Fixed potential integer overflow in env_sleep_msec() function in ThreadX environment implementation by rearranging calculation order in the sleep duration formula.

- Fixed CERT-C INT31-C violation in RPMsg-Lite where bitwise NOT operations on integer constants were performed in signed integer context before being cast to unsigned. This could potentially lead to misinterpreted data on imx943 platform.
- Added RL_MAX_BUFFER_COUNT (32768U) and RL_MAX_VRING_ALIGN (65536U) limit to ensure alignment values cannot contribute to integer overflow
- Fixed CERT INT31-C violation in vring_need_event(), added cast to uint16_t for each operand.

v5.1.4 - 27-Mar-2025

Added

- Add KW43B43 porting layer

Changed

- Doxygen bump to version 1.9.6

v5.1.3 - 13-Jan-2025

Added

- Memory cache management of shared memory. Enable with #define RL_USE_DCACHE (1) in rpmsg_config.h in case of data cache is used.
- Cmake/Kconfig support added.
- Porting layers for imx95, imxrt700, mcmxw71x, mcmxw72x, kw47b42 added.

v5.1.2 - 08-Jul-2024

Changed

- Zephyr-related changes.
- Minor Misra corrections.

v5.1.1 - 19-Jan-2024

Added

- Test suite provided.
- Zephyr support added.

Changed

- Minor changes in platform and env. layers, minor test code updates.

v5.1.0 - 02-Aug-2023

Added

- RPMsg-Lite: Added aarch64 support.

Changed

- RPMsg-Lite: Increased the queue size to (2 * RL_BUFFER_COUNT) to cover zero copy cases.

- Code formatting using LLVM16.

Fixed

- Resolved issues in ThreadX env. layer implementation.

v5.0.0 - 19-Jan-2023

Added

- Timeout parameter added to rpmsg_lite_wait_for_link_up API function.

Changed

- Improved debug check buffers implementation - instead of checking the pointer fits into shared memory check the presence in the VirtIO ring descriptors list.
- VRING_SIZE is set based on number of used buffers now (as calculated in vring_init) - updated for all platforms that are not communicating to Linux rpmsg counterpart.

Fixed

- Fixed wrong RL_VRING_OVERHEAD macro comment in platform.h files
- Misra corrections.

v4.0.0 - 20-Jun-2022

Added

- Added support for custom shared memory arrangement per the RPMsg_Lite instance.
- Introduced new rpmsg_lite_wait_for_link_up() API function - this allows to avoid using busy loops in rtos environments, GitHub PR #21.

Changed

- Adjusted rpmsg_lite_is_link_up() to return RL_TRUE/RL_FALSE.

v3.2.0 - 17-Jan-2022

Added

- Added support for i.MX8 MP multicore platform.

Changed

- Improved static allocations - allow OS-specific objects being allocated statically, GitHub PR #14.
- Aligned rpmsg_env_xos.c and some platform layers to latest static allocation support.

Fixed

- Minor Misra and typo corrections, GitHub PR #19, #20.

v3.1.2 - 16-Jul-2021

Added

- Addressed MISRA 21.6 rule violation in rpmsg_env.h (use SDK's PRINTF in MCUXpressoSDK examples, otherwise stdio printf is used).
- Added environment layers for XOS.
- Added support for i.MX RT500, i.MX RT1160 and i.MX RT1170 multicore platforms.

Fixed

- Fixed incorrect description of the rpmsg_lite_get_endpoint_from_addr function.

Changed

- Updated RL_BUFFER_COUNT documentation (issue #10).
- Updated imxrt600_hifi4 platform layer.

v3.1.1 - 15-Jan-2021

Added

- Introduced RL_ALLOW_CONSUMED_BUFFERS_NOTIFICATION config option to allow opposite side notification sending each time received buffers are consumed and put into the queue of available buffers.
- Added environment layers for Threadx.
- Added support for i.MX8QM multicore platform.

Changed

- Several MISRA C-2012 violations addressed.

v3.1.0 - 22-Jul-2020

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed (7.4).
- Fixed missing lock in rpmsg_lite_rx_callback() for QNX env.
- Correction of rpmsg_lite_instance structure members description.
- Address -Waddress-of-packed-member warnings in GCC9.

Changed

- Clang update to v10.0.0, code re-formatted.

v3.0.0 - 20-Dec-2019

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed, incl. data types consolidation.

- Code formatted.

v2.2.0 - 20-Mar-2019

Added

- Added configuration macro RL_DEBUG_CHECK_BUFFERS.
- Several MISRA violations fixed.
- Added environment layers for QNX and Zephyr.
- Allow environment context required for some environment (controlled by the RL_USE_ENVIRONMENT_CONTEXT configuration macro).
- Data types consolidation.

v1.1.0 - 28-Apr-2017

Added

- Supporting i.MX6SX and i.MX7D MPU platforms.
- Supporting LPC5411x MCU platform.
- Baremental and FreeRTOS support.
- Support of copy and zero-copy transfer.
- Support of static API (without dynamic allocations).

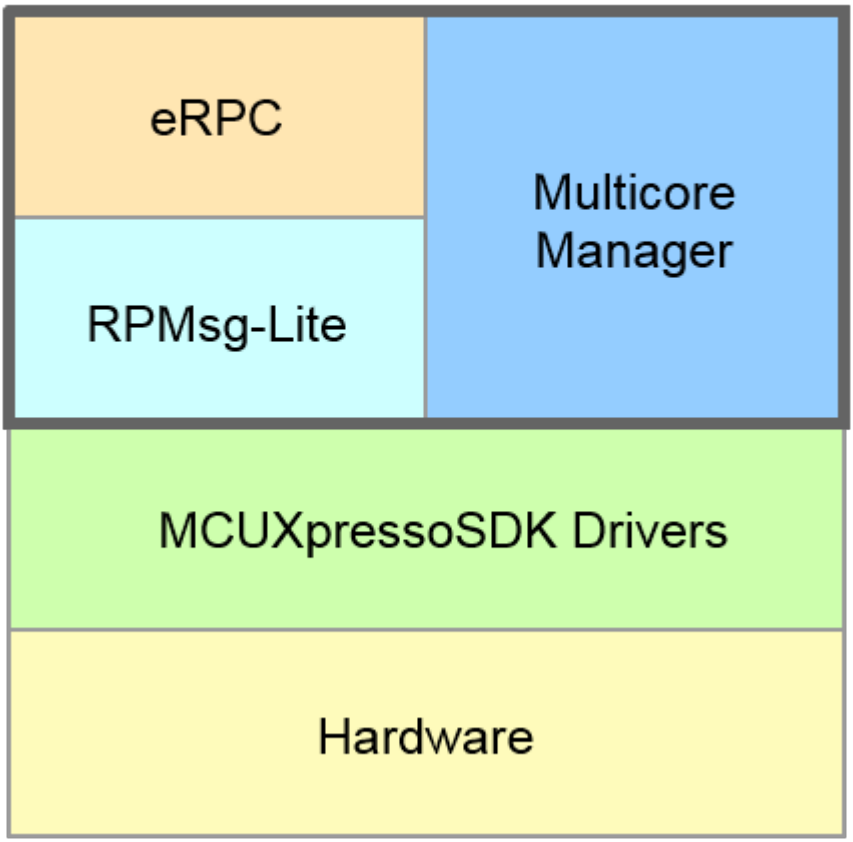
Multicore Manager (MCMGR)

Quick links

- *Changelog*
- API documentation (Doxygen): https://nxp-mcuxpresso.github.io/mcux-mcmgr/

The Multicore Manager (MCMGR) software library provides a number of services for multicore systems. This library is distributed as a part of the Multicore SDK (MCSDK). Together, the MCSDK and the MCUXpresso SDK (SDK) form a framework for development of software for NXP multicore devices.

The MCMGR component is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr directory.



The Multicore Manager provides the following major functions:

- Maintains information about all cores in system.
- Secondary/auxiliary core(s) startup and shutdown.
- Remote core monitoring and event handling.

Usage of the MCMGR software component

The main use case of MCMGR is the secondary/auxiliary core start. This functionality is performed by the public API function.

Example of MCMGR usage to start secondary core:

```
#include "mcmgr.h"

void main()
{
    /* Initialize MCMGR, low level multicore management library */
    MCMGR_Init();

    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass "1" as startup data,␣
↪starting synchronously. */
    MCMGR_StartCore(kMCMGR_Core1, CORE1_BOOT_ADDRESS, 1, kMCMGR_Start_Synchronous);
```

(continues on next page)

(continued from previous page)

```
.
.
.
    /* Stop secondary core execution. */
    MCMGR_StopCore(kMCMGR_Core1);
}
```

Some platforms allow stopping and re-starting the secondary core application again, using the MCMGR_StopCore / MCMGR_StartCore API calls. It is necessary to ensure the initially loaded image is not corrupted before re-starting, especially if it deals with the RAM target. Cache coherence has to be considered/ensured as well.

It could also happen that the secondary core application stops running correctly and the primary core application does not know about that situation. Therefore, it is beneficial to implement a mechanism for core health monitoring. The test_heartbeat unit test can serve as an example how to ensure that: secondary core could periodically send heartbeat signals to the primary core using MCMGR_TriggerEvent() API to indicate that it is alive and functioning properly.

Another important MCMGR feature is the ability for remote core monitoring and handling of events such as reset, exception, and application events. Application-specific callback functions for events are registered by the MCMGR_RegisterEvent() API. Triggering these events is done using the MCMGR_TriggerEvent() API. mcmgr_event_type_t enums all possible event types.

An example of MCMGR usage for remote core monitoring and event handling. Code for the primary side:

```
#include "mcmgr.h"

#define APP_RPMSG_READY_EVENT_DATA  (1)
#define APP_NUMBER_OF_CORES (2)
#define APP_SECONDARY_CORE kMCMGR_Core1

/* Callback function registered via the MCMGR_RegisterEvent() and triggered by MCMGR_TriggerEvent()␣
↪called on the secondary core side */
void RPMsgRemoteReadyEventHandler(mcmgr_core_t coreNum, uint16_t eventData, void *context)
{
    uint16_t *data = &((uint16_t *)context)[coreNum];

    *data = eventData;
}

void main()
{
    uint16_t RPMsgRemoteReadyEventData[NUMBER_OF_CORES] = {0};

    /* Initialize MCMGR, low level multicore management library */
    MCMGR_Init();

    /* Register the application event before starting the secondary core */
    MCMGR_RegisterEvent(kMCMGR_RemoteApplicationEvent, RPMsgRemoteReadyEventHandler, (void␣
↪*)RPMsgRemoteReadyEventData);

    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass rpmsg_lite_base address␣
↪as startup data, starting synchronously. */
    MCMGR_StartCore(APP_SECONDARY_CORE, CORE1_BOOT_ADDRESS, (uint32_t)rpmsg_lite_
↪base, kMCMGR_Start_Synchronous);

    /* Wait until the secondary core application signals the rpmsg remote has been initialized and is ready to␣
↪communicate. */
    while(APP_RPMSG_READY_EVENT_DATA != RPMsgRemoteReadyEventData[APP_SECONDARY_
↪CORE]) {};
```

(continues on next page)

(continued from previous page)

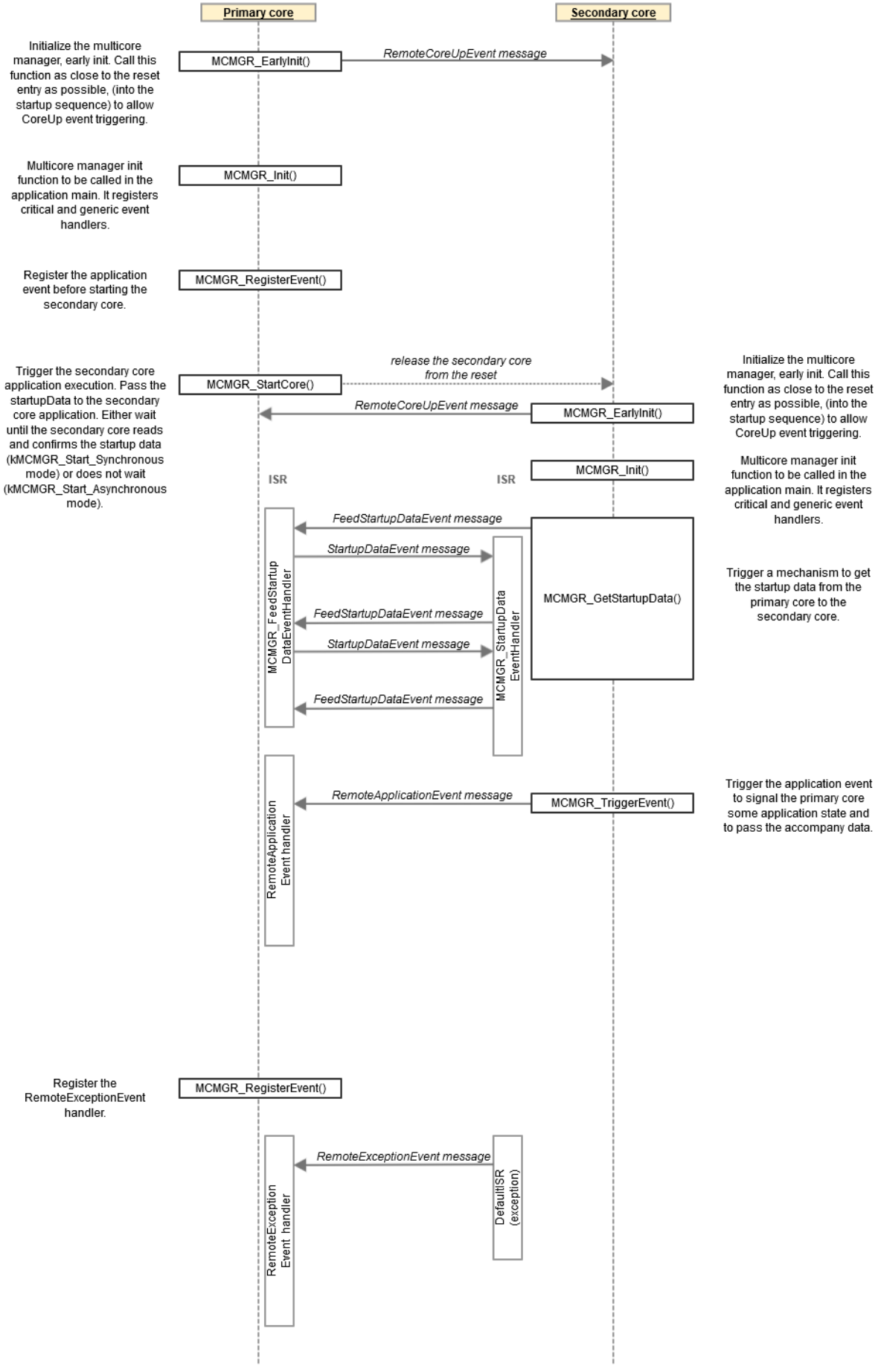
```
.
.
.
}
```

Code for the secondary side:

```
#include "mcmgr.h"

#define APP_RPMSG_READY_EVENT_DATA  (1)

void main()
{
    /* Initialize MCMGR, low level multicore management library */
    MCMGR_Init();
.
.
.

    /* Signal the to other core that we are ready by triggering the event and passing the APP_RPMSG_
↪READY_EVENT_DATA */
    MCMGR_TriggerEvent(kMCMGR_Core0, kMCMGR_RemoteApplicationEvent, APP_RPMSG_
↪READY_EVENT_DATA);
.
.
.
}
```

MCMGR Data Exchange Diagram

The following picture shows how the handshakes are supposed to work between the two cores in the MCMGR software.


Primary core
Secondary core
Initialize the multicore manager, early init. Call this function as close to the reset entry as possible, (into the startup sequence) to allow CoreUp event triggering.
MCMGR_EarlyInit()
RemoteCoreUpEvent message
Multicore manager init function to be called in the application main. It registers critical and generic event handlers.
MCMGR_Init()
Register the application event before starting the secondary core.
MCMGR_RegisterEvent()
Trigger the secondary core application execution. Pass the startupData to the secondary core application. Either wait until the secondary core reads and confirms the startup data (kMCMGR_Start_Synchronous mode) or does not wait (kMCMGR_Start_Asynchronous mode).
MCMGR_StartCore()
release the secondary core from the reset
RemoteCoreUpEvent message
MCMGR_EarlyInit()
Initialize the multicore manager, early init. Call this function as close to the reset entry as possible, (into the startup sequence) to allow CoreUp event triggering.
MCMGR_Init()
Multicore manager init function to be called in the application main. It registers critical and generic event handlers.
ISR
ISR
FeedStartupDataEvent message
StartupDataEvent message
FeedStartupDataEvent message
StartupDataEvent message
FeedStartupDataEvent message
MCMGR_FeedStartup DataEventHandler
MCMGR_StartupData EventHandler
MCMGR_GetStartupData()
Trigger a mechanism to get the startup data from the primary core to the secondary core.
RemoteApplicationEvent message
MCMGR_TriggerEvent()
RemoteApplication Event handler
Trigger the application event to signal the primary core some application state and to pass the accompany data.
Register the RemoteExceptionEvent handler.
MCMGR_RegisterEvent()
RemoteExceptionEvent message
RemoteException Event handler
DefaultISR (exception)

Changelog Multicore Manager

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

v5.3.0

Added

- Added MCXE32B (dual Cortex-M7) porting layer.
- Remove obsolete unused platform layers
- Added initialization guard to public MCMGR API functions (RPSDK-1183): guarded functions return kStatus_MCMGR_NotReady before MCMGR_Init() completes.

Fixed

- MCUX-88924: Fix KW43 ICS init-handshake hang (RPSDK-1183 regression): s_mcmgrInitialized is now set before mcmgr_platform_init_internal() enables the MU RX IRQ, so re-entrant ISR dispatch during MCMGR_Init() is no longer silently dropped by MCMGR_CHECK_INIT().
- MCUX-89876: Fix Coverity DEADCODE and UNUSED_VALUE findings in mcmgr_internal_core_api_imxrt700.c: align mcmgr_mu_channel_handler event-range guard with positive-logic form (matching imxrt1170), and aggregate mcmgr_trigger_event_internal return values in mcmgr_platform_init_internal_early so earlier errors are not silently overwritten.

v5.2.0

Changed

- Merged internal platform initialization into a single $\mathrm{mcmgr_platform_init_internal()}$ function.
- Simplified initialization API: $\mathrm{MCMGR_Init()}$ now handles both early and standard initialization internally.

Deprecated

- $\mathrm{MCMGR_EarlyInit()}$ is now deprecated and a no-op. It remains for backward compatibility and may be removed in a future major version. Replace calls to $\mathrm{MCMGR_EarlyInit()}$ + $\mathrm{MCMGR_Init()}$ with a single $\mathrm{MCMGR_Init()}$ call.

Migration Guide

Before:

```
MCMGR_EarlyInit();
// ...
MCMGR_Init();
```

After:

```
MCMGR_Init();
```

v5.1.0

Added

- Added mcxl14x parts support
- Added mcxw70 parts support
- Add new config MCMGR_REMOTE_APP_EVENT_COUNT for configurable remote application event count. This will expand number of application events available for MCMGR users. kMCMGR_RemoteApplicationEvent, kMCMGR_RemoteApplicationEvent1 .. kMCMGR_RemoteApplicationEvent4.
- Align Kinetis parts porting to changed register map

Fixed

- Fix MISRA C-2012 Rule 20.8 violations.

v5.0.2

Added

- Added gcov options and configs to support mcmgr code coverage
- Added new test_weak_mu_isr testcase for devices with MU peripheral
- Added new test_heartbeat testcase showing heartbeat mechanism between primary and secondary cores using the MCMGR

v5.0.1

Added

- Added frdmimxrt1186 unit testing

Changed

- [KW43] Rename core#1 reset control register

Fixed

- Added CX flag into CMakeLists.txt to allow c++ build compatibility.
- Fix path to mcmgr headers directory in doxyfile

v5.0.0

Added

- Added MCMGR_BUSY_POLL_COUNT macro to prevent infinite polling loops in MCMGR operations.
- Implemented timeout mechanism for all polling loops in MCMGR code.
- Added support to handle more then two cores. Breaking API change by adding parameter coreNum specifying core number in functions bellow.
  - MCMGR_GetStartupData(uint32_t *startupData, mcmgr_core_t coreNum)
  - MCMGR_TriggerEvent(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)
  - MCMGR_TriggerEventForce(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)
  - typedef void (*mcmgr_event_callback_t)(uint16_t data, void *context, mcmgr_core_t coreNum);

When registering the event with function MCMGR_RegisterEvent() user now needs to provide callbackData pointer to array of elements per every core in system (see README.md for example).In case of systems with only two cores the coreNum in callback can be ignored as events can arrive only from one core. Please see Porting guide for more details: Porting-GuideTo_v5.md

- Updated all porting files to support new MCMGR API.
- Added new platform specific include file mcmgr_platform.h. It will contain common platform specific macros that can be then used in mcmgr and application. e.g. platform core count MCMGR_CORECOUNT 4.
- Move all header files to new inc directory.
- Added new platform-specific include files inc/platform/<platform_name>/mcmgr_platform.h.

Added

- Add MCXL20 porting layer and unit testing

v4.1.7

Fixed

- mcmgr_stop_core_internal() function now returns kStatus_MCMGR_NotImplemented status code instead of kStatus_MCMGR_Success when device does not support stop of secondary core. Ports affected: kw32w1, kw45b41, kw45b42, mcxw716, mcxw727.

[v4.1.6]

Added

- Multicore Manager moved to standalone repository.
- Add porting layers for imxrt700, mcmxw727, kw47b42.
- New MCMGR_ProcessDeferredRxIsr() API added.

[v4.1.5]

Added

- Add notification into MCMGR_EarlyInit and mcmgr_early_init_internal functions to avoid using uninitialized data in their implementations.

[v4.1.4]

Fixed

- Avoid calling tx isr callbacks when respective Messaging Unit Transmit Interrupt Enable flag is not set in the CR/TCR register.
- Messaging Unit RX and status registers are cleared after the initialization.

[v4.1.3]

Added

- Add porting layers for imxrt1180.

Fixed

- mu_isr() updated to avoid calling tx isr callbacks when respective Transmit Interrupt Enable flag is not set in the CR/TCR register.
- mcmgr_mu_internal.c code adaptation to new supported SoCs.

[v4.1.2]

Fixed

- Update mcmgr_stop_core_internal() implementations to set core state to kMCMGR_ResetCoreState.

[v4.1.0]

Fixed

- Code adjustments to address MISRA C-2012 Rules

[v4.0.3]

Fixed

- Documentation updated to describe handshaking in a graphic form.
- Minor code adjustments based on static analysis tool findings

[v4.0.2]

Fixed

- Align porting layers to the updated MCUXpressoSDK feature files.

[v4.0.1]

Fixed

- Code formatting, removed unused code

[v4.0.0]

Added

- Add new MCMGR_TriggerEventForce() API.

[v3.0.0]

Removed

- Removed MCMGR_LoadApp(), MCMGR_MapAddress() and MCMGR_SignalReady()

Modified

- Modified MCMGR_GetStartupData()

Added

- Added MCMGR_EarlyInit(), MCMGR_RegisterEvent() and MCMGR_TriggerEvent()
- Added the ability for remote core monitoring and event handling

[v2.0.1]

Fixed

- Updated to be Misra compliant.

[v2.0.0]

Added

- Support for lpcxpresso54114 board.

[v1.1.0]

Fixed

- Ported to KSDK 2.0.0.

[v1.0.0]

Added

- Initial release.

eRPC

eRPC (Embedded RPC) is an open-source remote procedure call (RPC) system suitable for multi-core and multiprocessor embedded systems.

Quick links

- *Getting Started*
- *Changelog*
- Upstream project: https://github.com/EmbeddedRPC/erpc

eRPC Getting Started

Overview

This *Getting Started User Guide* shows software developers how to use Remote Procedure Calls (RPC) in embedded multicore microcontrollers (eRPC).

The eRPC documentation is located in the *<MCUXpressoSDK_install_dir>/ middleware/multicore/erpc/doc* folder.

Create an eRPC application

This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.

2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
    1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
    2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
    3. Add infrastructure files.
    4. Add user code for client and server applications.
    5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

Multicore server application

The “Matrix multiply” eRPC server project is located in the following folder:

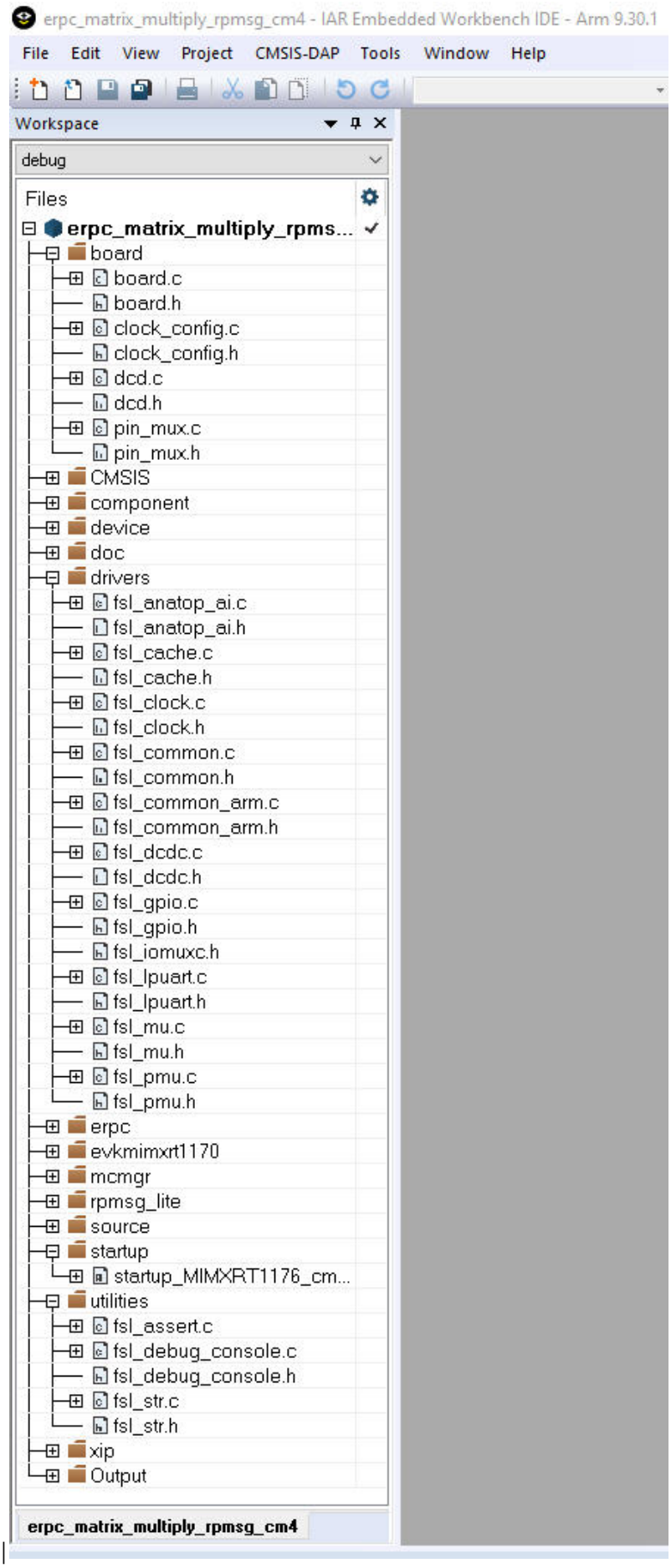
<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar

The project files for the eRPC server have the $_\mathrm{cm4}$ suffix.

Server project basic source files

The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- *<MCUXpressoSDK_install_dir>/devices/<device>*
- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

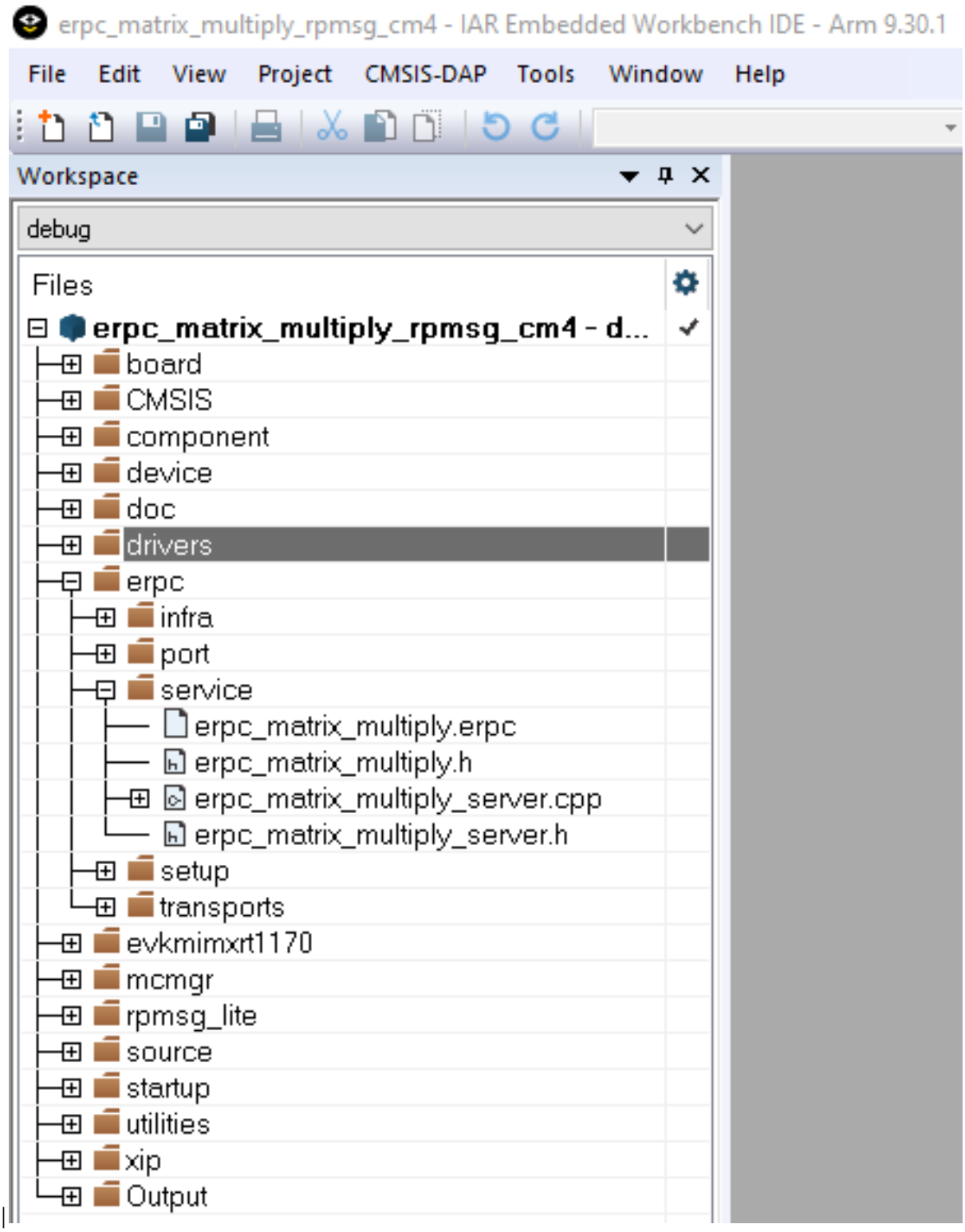
Server related generated files

The server-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/s



Parent topic: Multicore server application

Server infrastructure files

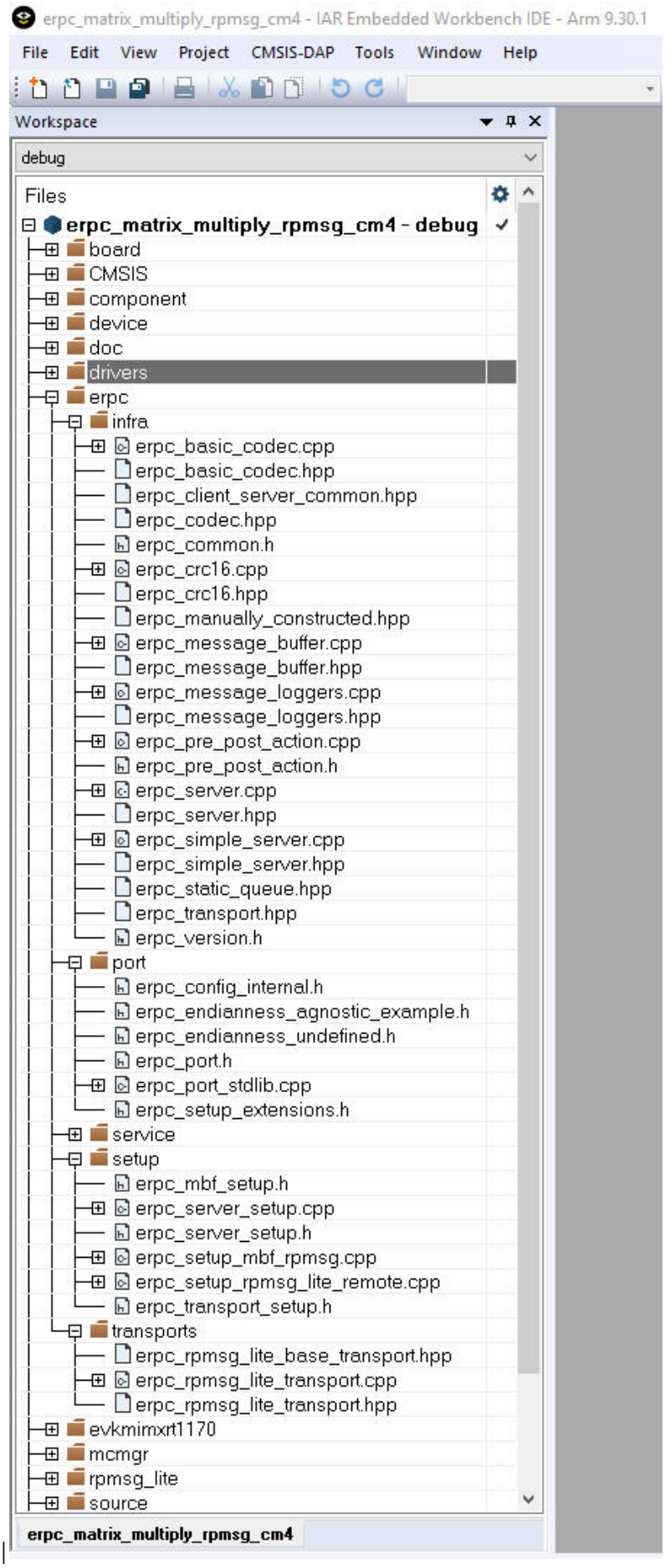
The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the *C/C++*

language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, erpc_server.hpp, erpc_server.cpp, erpc_simple_server.hpp, and erpc_simple_server.cpp, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The erpc_common.hpp file is used for common eRPC definitions, typedefs, and enums.
  - The erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: erpc_message_buffer.h and erpc_message_buffer.cpp.
  - The erpc_transport.h file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
  - erpc_port_stdlib.cpp file ensures adaptation to stdlib.
  - erpc_config_internal.h internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The erpc_server_setup.h and erpc_server_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
  - The erpc_transport_setup.h and erpc_setup_rpmsg_lite_remote.cpp files needs to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files need to be added into the server project.


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_server.cpp
erpc_server.hpp
erpc_simple_server.cpp
erpc_simple_server.hpp
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_mbf_setup.h
erpc_server_setup.cpp
erpc_server_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_remote.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

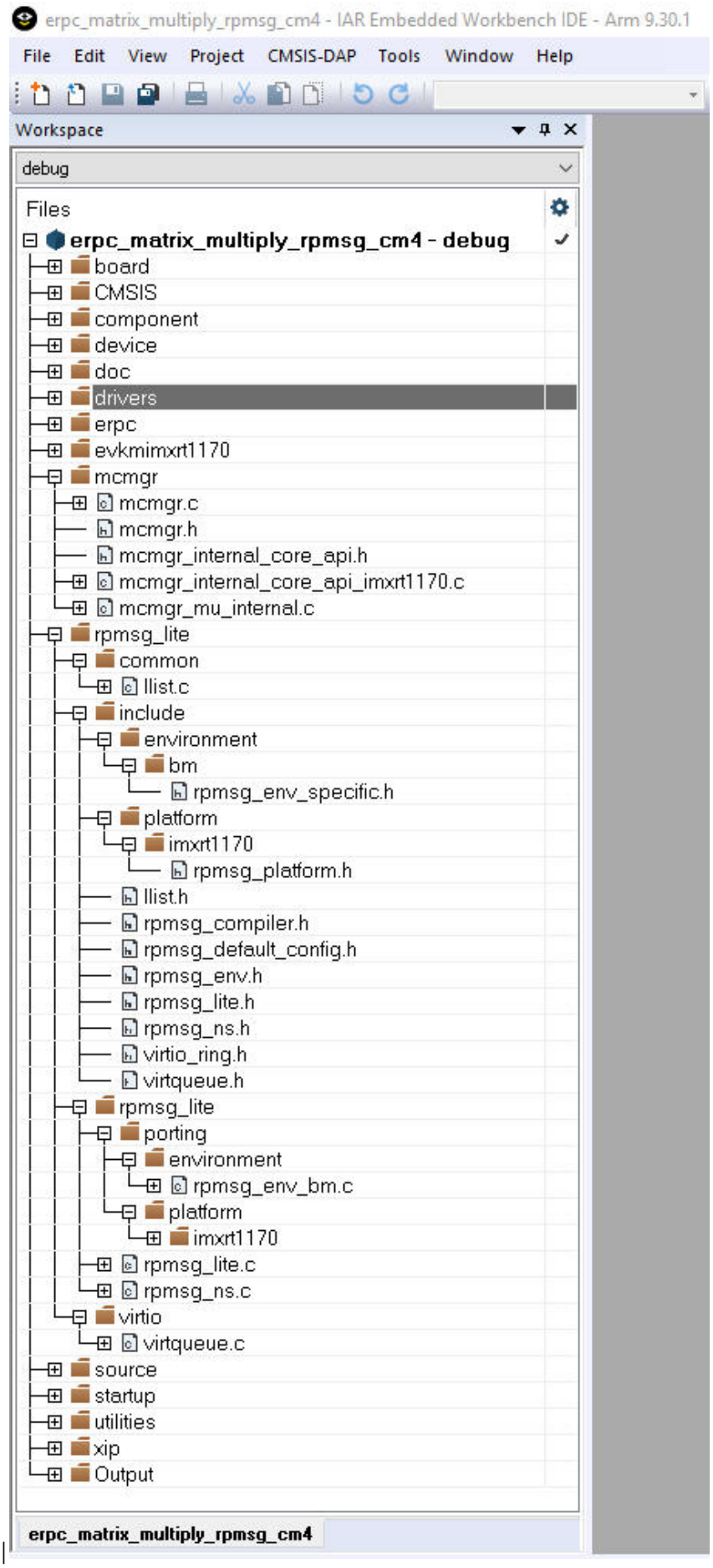
Server multicore infrastructure files

Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

Server user code

The server's user code is stored in the main_core1.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4

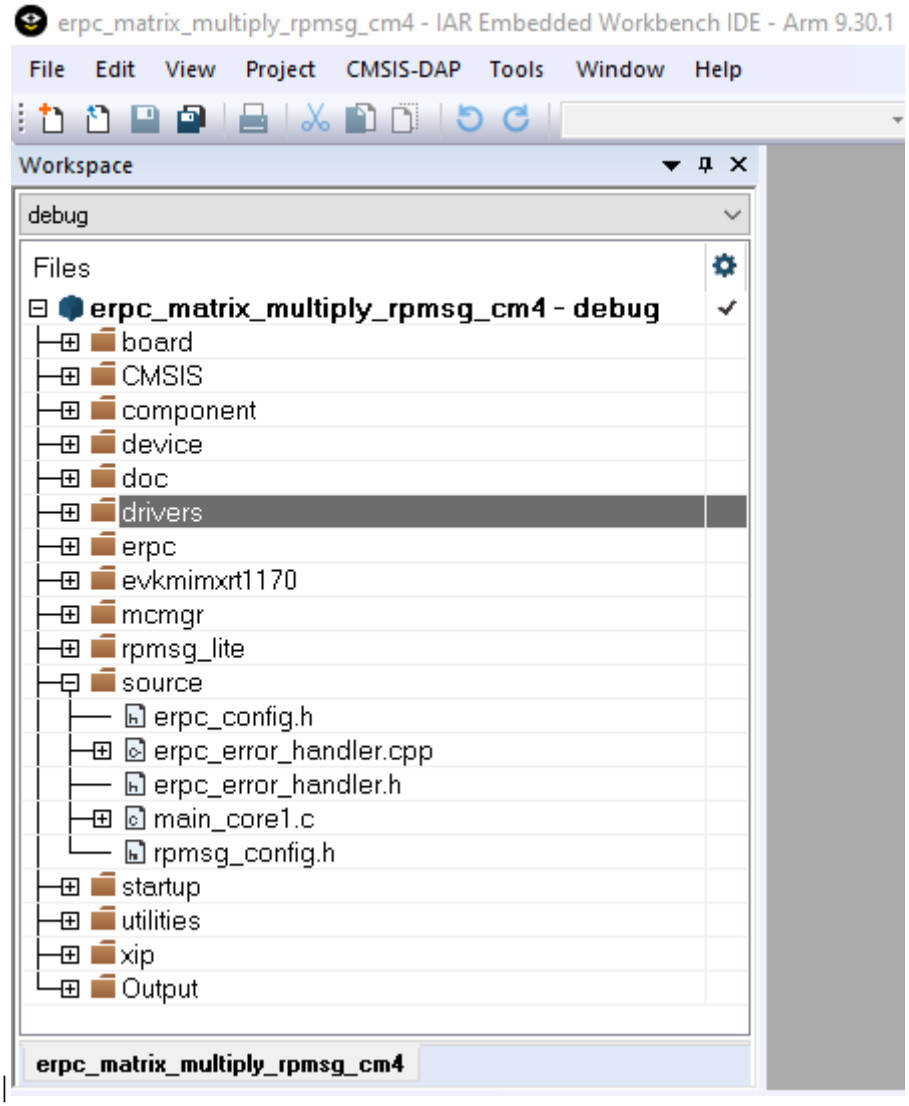
The main_core1.c file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the erpc_error_handler.h and erpc_error_handler.cpp files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
  ...
}
int main()
{
  ...
  /* RPMsg-Lite transport layer initialization */
  erpc_transport_t transport;
  transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
  ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_rpmsg_init(transport);
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server);
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg* folder.



Parent topic: Multicore server application

Parent topic:*Create an eRPC application*

Multicore client application

The “Matrix multiply” eRPC client project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar

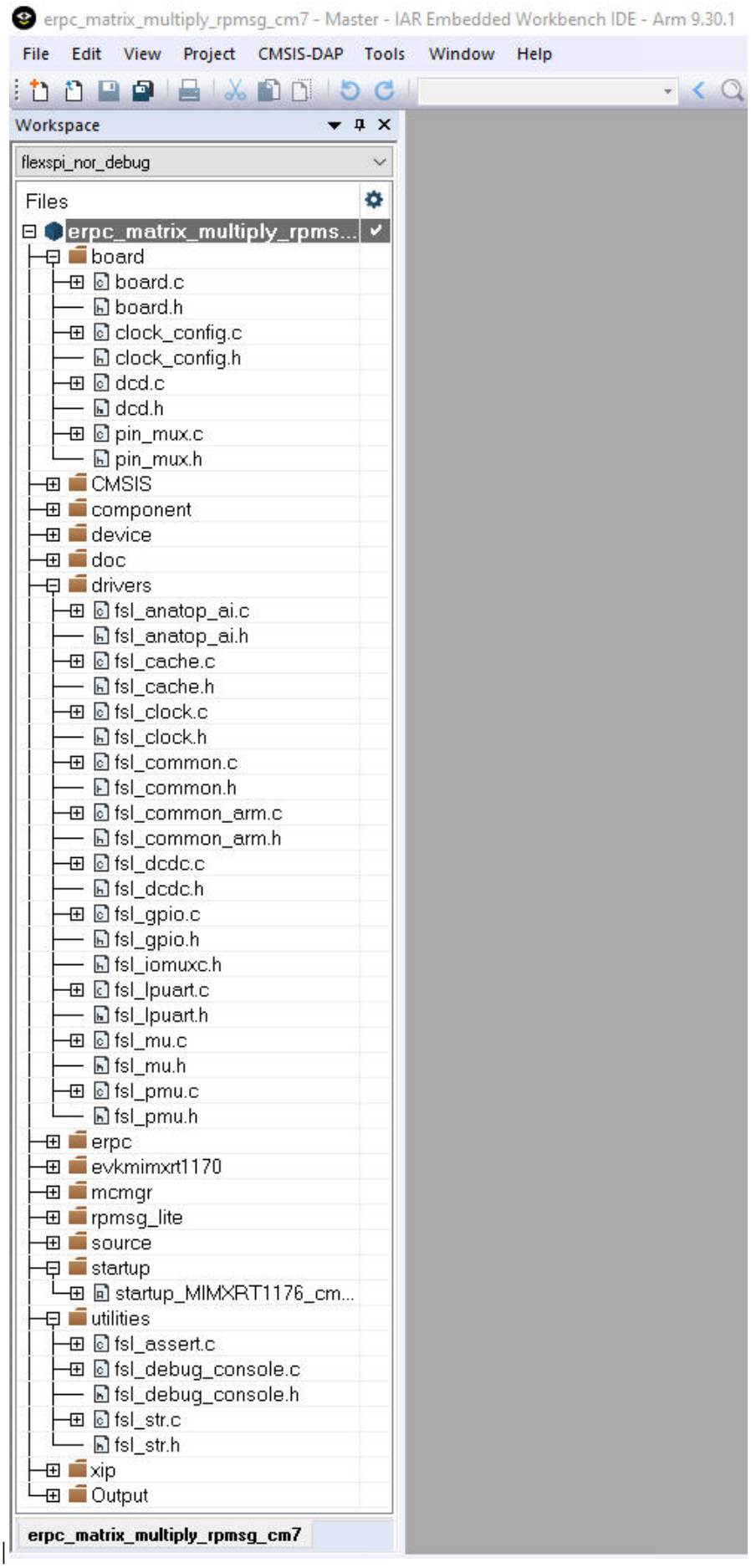
Project files for the eRPC client have the _cm7 suffix.

Client project basic source files

The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- *<MCUXpressoSDK_install_dir>/devices/<device>*

- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

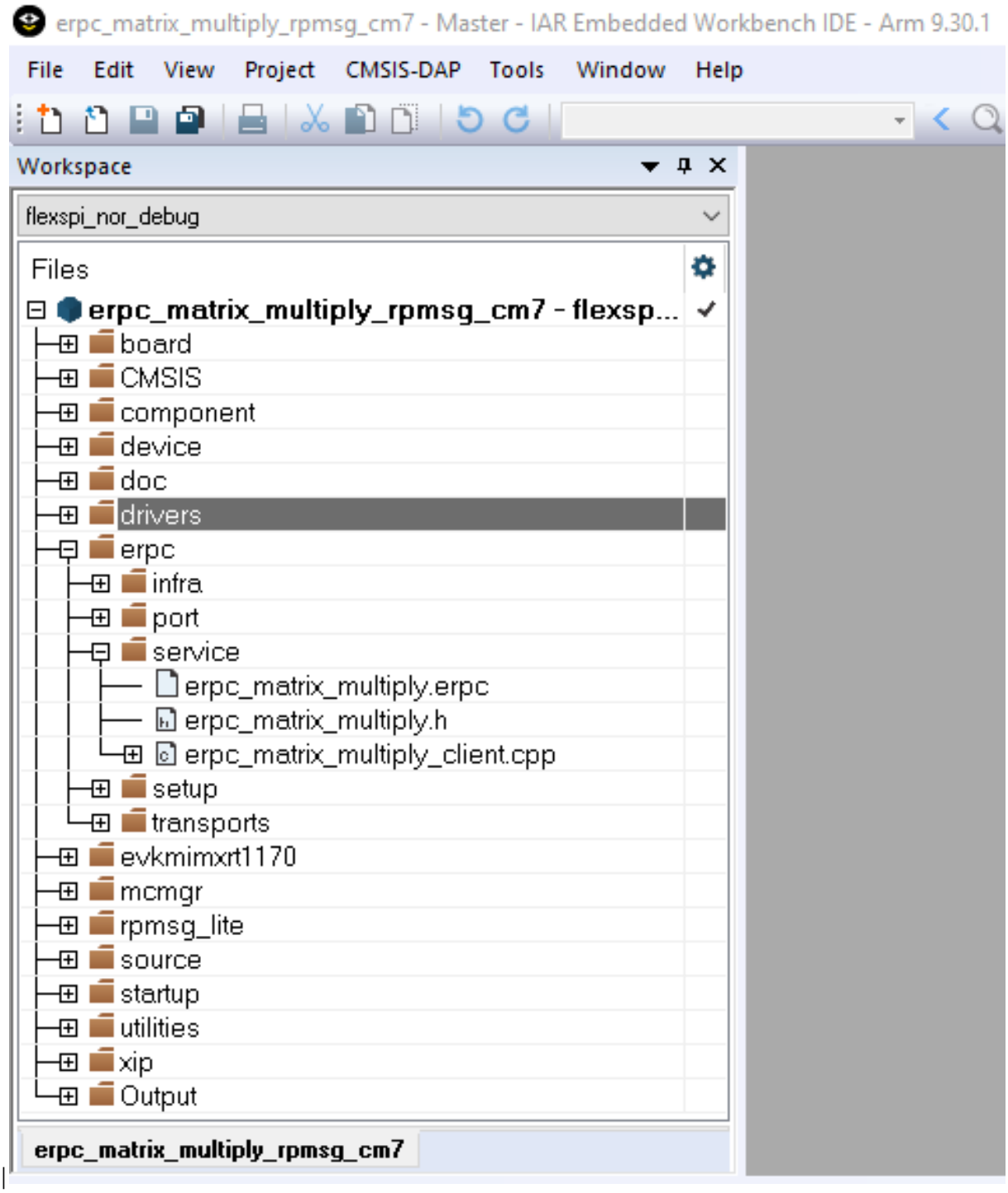
Parent topic: Multicore client application

Client-related generated files

The client-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/* folder.



Parent topic: Multicore client application

Client infrastructure files

The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, erpc_client_manager.h and erpc_client_manager.cpp, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- erpc_common.h file is used for common eRPC definitions, typedefs, and enums.
- erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: erpc_message_buffer.hpp and erpc_message_buffer.cpp.
- erpc_transport.hpp file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

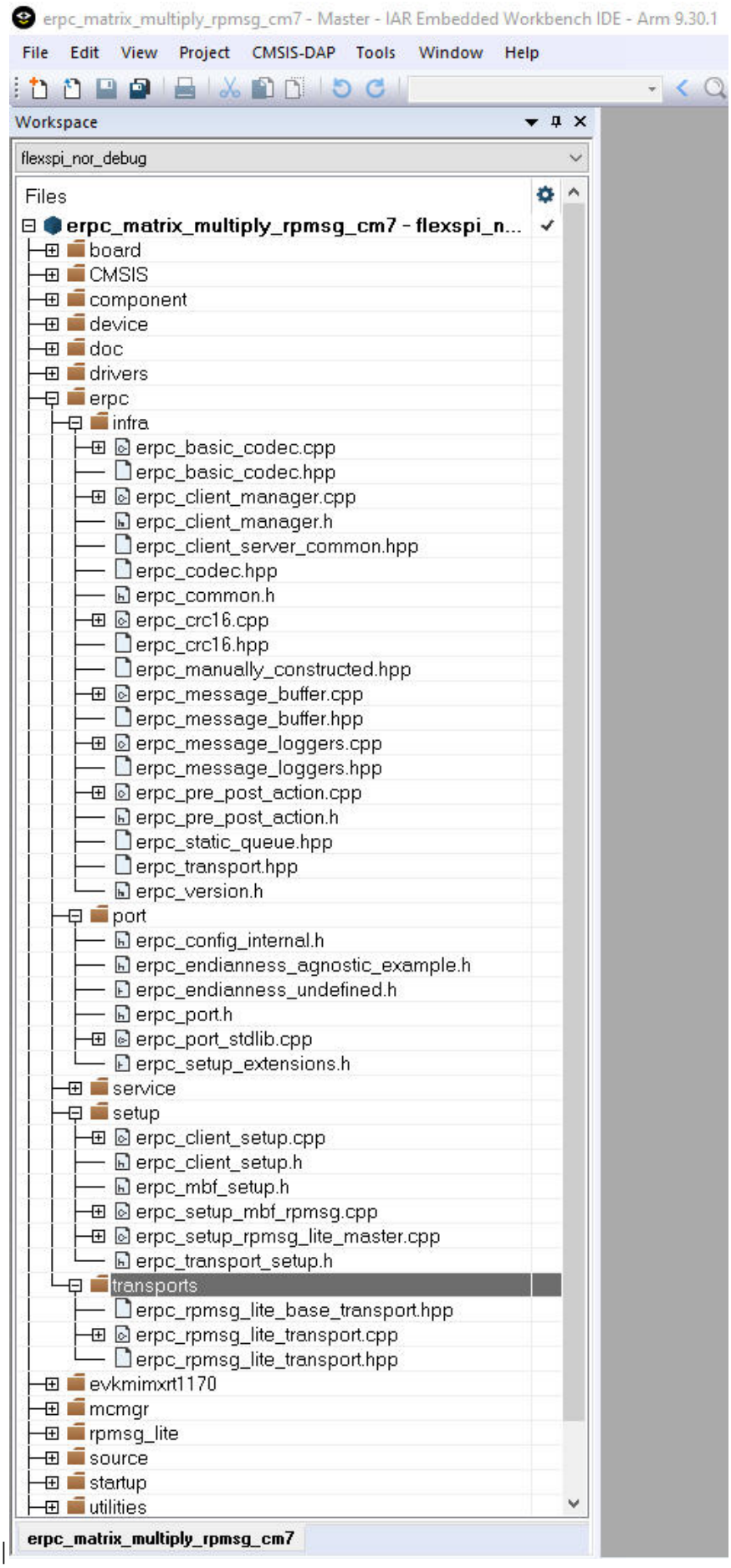
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- erpc_client_setup.h and erpc_client_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
- erpc_transport_setup.h and erpc_setup_rpmsg_lite_master.cpp files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files needs to be added into the client project.


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_n...
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_manager.cpp
erpc_client_manager.h
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_client_setup.cpp
erpc_client_setup.h
erpc_mbf_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_master.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
utilities
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

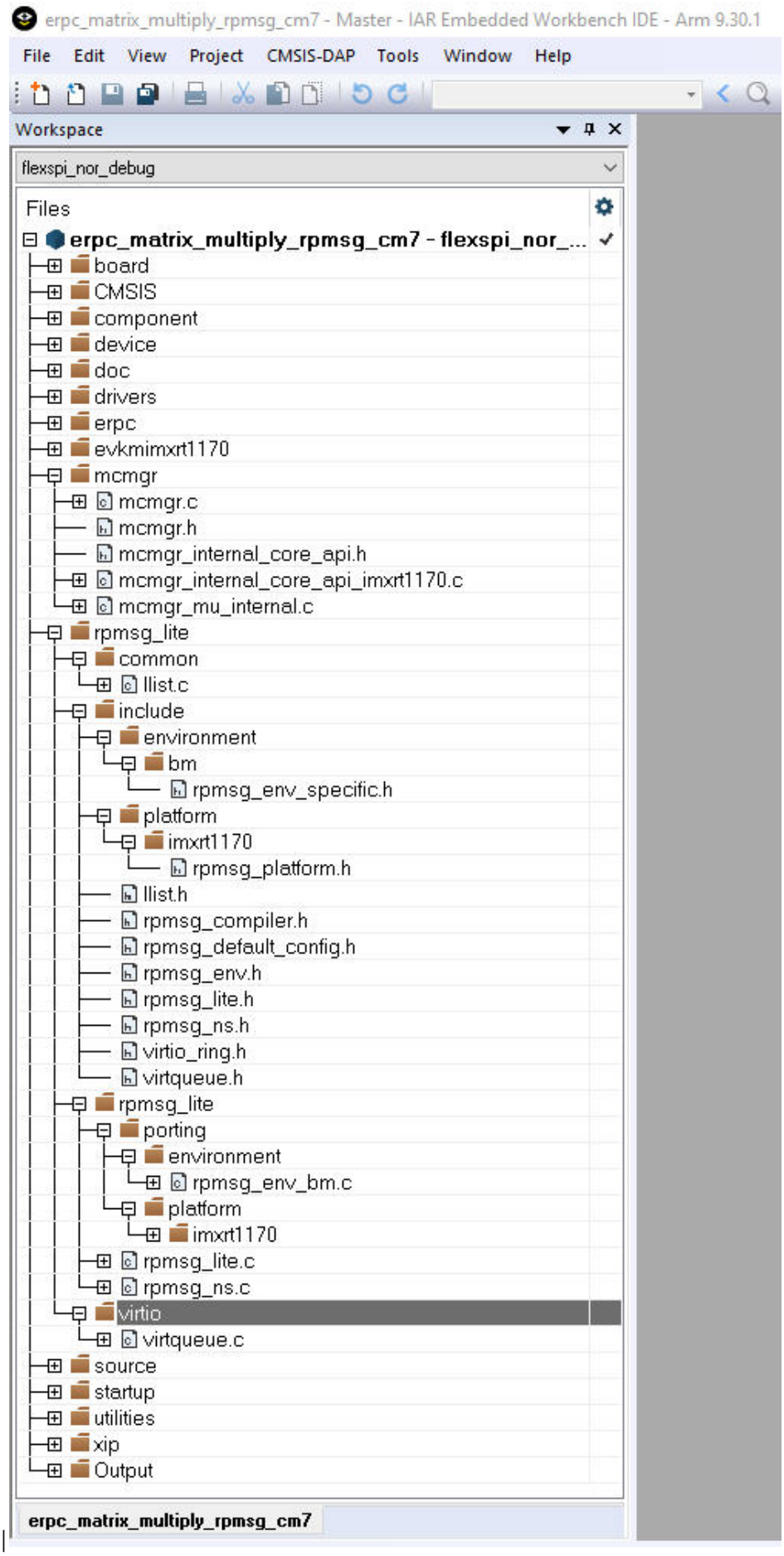
Client multicore infrastructure files

Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_nor_...
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

Client user code

The client's user code is stored in the main_core0.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_example/erpc_matrix_multiply_rpmsg/cm7

The main_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc_error_handler.h and erpc_error_handler.cpp files.

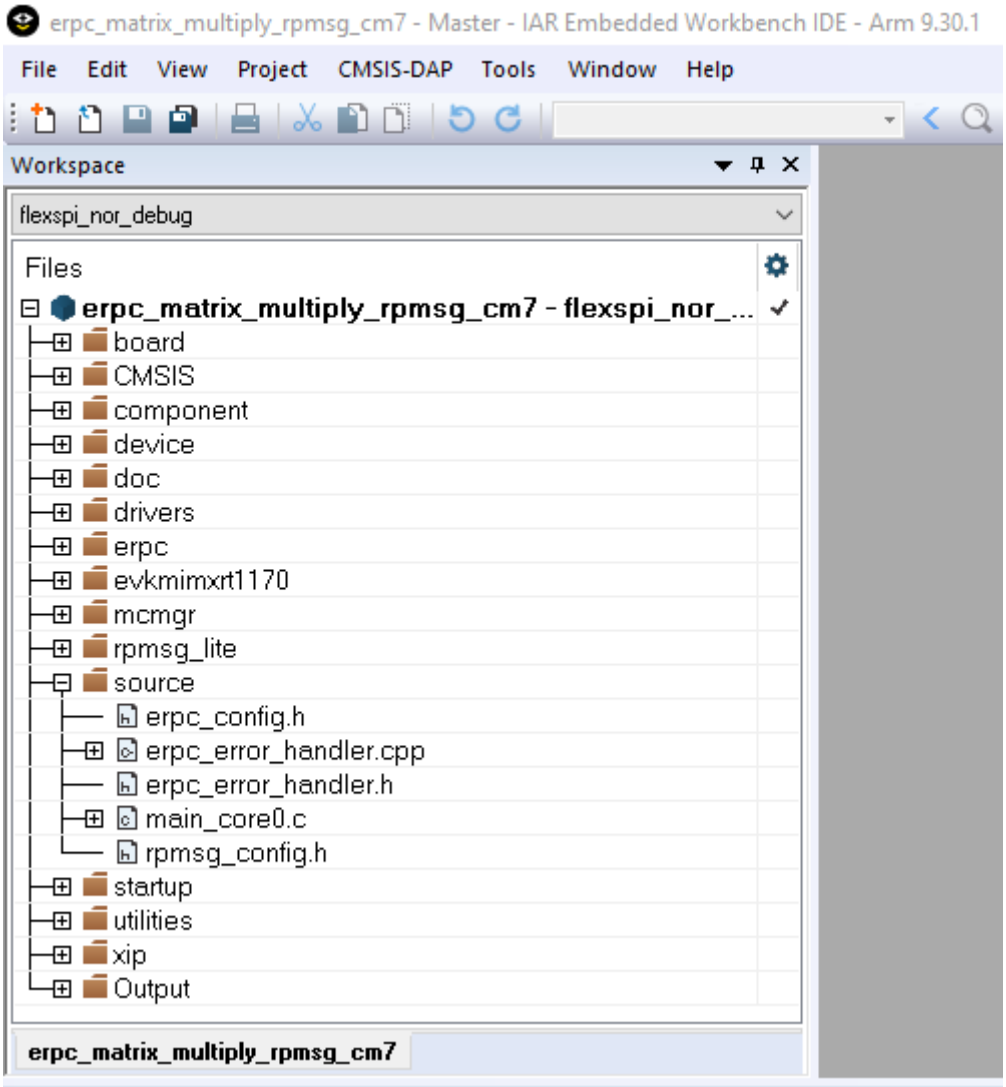
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg



|

|

Parent topic: Multicore client application

Parent topic:*Create an eRPC application*

Multiprocessor server application

The “Matrix multiply” eRPC server project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir»/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Server user code

The server's user code is stored in the main_server.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_lay* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
  ...
}
int main()
{
  ...
  /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
  erpc_transport_t transport;
  transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_dynamic_init();
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server)
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Parent topic: Multiprocessor server application

Multiprocessor client application

The "Matrix multiply" eRPC client project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<tr* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.hpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Client user code

The client's user code is stored in the main_client.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

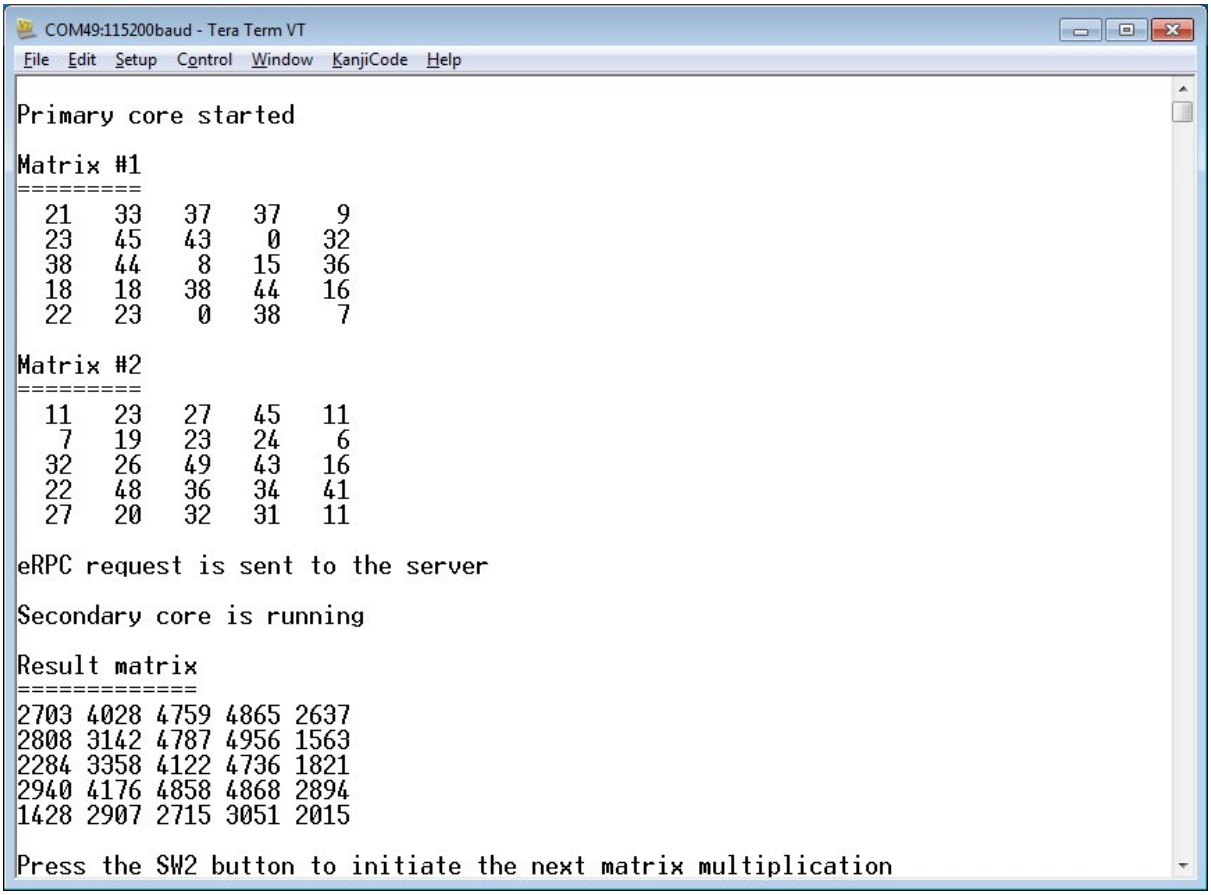
Parent topic: Multiprocessor client application

Parent topic: Multiprocessor server application

Parent topic:*Create an eRPC application*

Running the eRPC application

Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the <*MCUXpressoSDK_install_dir*>/*docs* folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:*Create an eRPC application*

Parent topic:*eRPC example*

eRPC example

This section shows how to create an example eRPC application called “Matrix multiply”, which implements one eRPC function (matrix multiply) with two function parameters (two matrices). The client-side application calls this eRPC function, and the server side performs the multiplication of received matrices. The server side then returns the result.

For example, use the NXP MIMXRT1170-EVK board as the target dual-core platform, and the IAR Embedded Workbench for ARM (EWARM) as the target IDE for developing the eRPC example.

- The primary core (CM7) runs the eRPC client.
- The secondary core (CM4) runs the eRPC server.
- RPMsg-Lite (Remote Processor Messaging Lite) is used as the eRPC transport layer.

The “Matrix multiply” application can be also run in the multi-processor setup. In other words, the eRPC client running on one SoC comunicates with the eRPC server that runs on anothe SoC, utilizing different transport channels. It is possible to run the board-to-PC example (PC as the eRPC server and a board as the eRPC client, and vice versa) and also the board-to-board example. These multiprocessor examples are prepared for selected boards only.

|Multicore application source and project files|*<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore*
|Multiprocessor application source and project files|*<MCUXpressoSDK_install_dir>/boards/<board_name>/multi*

<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<tr

| |eRPC source files|*<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/*| |RPMsg-Lite source files|*<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/*|

Designing the eRPC application

The matrix multiply application is based on calling single eRPC function that takes 2 two-dimensional arrays as input and returns matrix multiplication results as another 2 two-dimensional array. The IDL file syntax supports arrays with the dimension length set by the number only (in the current eRPC implementation). Because of this, a variable is declared in the IDL dedicated to store information about matrix dimension length, and to allow easy maintenance of the user and server code.

For a simple use of the two-dimensional array, the alias name (new type definition) for this data type has is declared in the IDL. Declaring this alias name ensures that the same data type can be used across the client and server applications.

Parent topic:*eRPC example*

Creating the IDL file

The created IDL file is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/s

The created IDL file contains the following code:

```
program erpc_matrix_multiply
/*! This const defines the matrix size. The value has to be the same as the
Matrix array dimension. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
const int32 matrix_size = 5;
/*! This is the matrix array type. The dimension has to be the same as the
matrix size const. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
type Matrix = int32[matrix_size][matrix_size];
interface MatrixMultiplyService {
erpcMatrixMultiply(in Matrix matrix1, in Matrix matrix2, out Matrix result_matrix) ->
void
}
```

Details:

- The IDL file starts with the program name (*erpc_matrix_multiply*), and this program name is used in the naming of all generated outputs.
- The declaration and definition of the constant variable named *matrix_size* follows next. The *matrix_size* variable is used for passing information about the length of matrix dimensions to the client/server user code.
- The alias name for the two-dimensional array type (*Matrix*) is declared.
- The interface group *MatrixMultiplyService* is located at the end of the IDL file. This interface group contains only one function declaration *erpcMatrixMultiply*.

- As shown above, the function's declaration contains three parameters of Matrix type: *matrix1* and *matrix2* are input parameters, while *result_matrix* is the output parameter. Additionally, the returned data type is declared as void.

When writing the IDL file, the following order of items is recommended:

1. Program name at the top of the IDL file.
2. New data types and constants declarations.
3. Declarations of interfaces and functions at the end of the IDL file.

Parent topic:*eRPC example*

Using the eRPC generator tool

|Windows OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Windows|
|Linux OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x64

<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x86

| |Mac OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Mac|

The files for the "Matrix multiply" example are pre-generated and already a part of the application projects. The following section describes how they have been created.

- The easiest way to create the shim code is to copy the erpcgen application to the same folder where the IDL file (*.erpc) is located; then run the following command:

  erpcgen <IDL_file>.erpc

- In the "Matrix multiply" example, the command should look like:

  erpcgen erpc_matrix_multiply.erpc

Additionally, another method to create the shim code is to execute the eRPC application using input commands:

- "-?"/"—help" – Shows supported commands.
- "-o <filePath>"/"—output<filePath>" – Sets the output directory.

For example,

```
<path_to_erpcgen>/erpcgen –o <path_to_output>
<path_to_IDL>/<IDL_file_name>.erpc
```

For the "Matrix multiply" example, when the command is executed from the default erpcgen location, it looks like:

erpcgen –o

../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service
../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/erpc_matrix_mu

In both cases, the following four files are generated into the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service folder.*

- erpc_matrix_multiply.h
- erpc_matrix_multiply_client.cpp
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

For multiprocessor examples, the eRPC file and pre-generated files can be found in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_common/erpc_matrix_multiply/serv* folder.

For Linux OS users:

- Do not forget to set the permissions for the eRPC generator application.
- Run the application as ./erpcgen... instead of as erpcgen ....

Parent topic:*eRPC example*

Create an eRPC application

This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
    1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
    2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
    3. Add infrastructure files.
    4. Add user code for client and server applications.
    5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

Multicore server application

The “Matrix multiply” eRPC server project is located in the following folder:

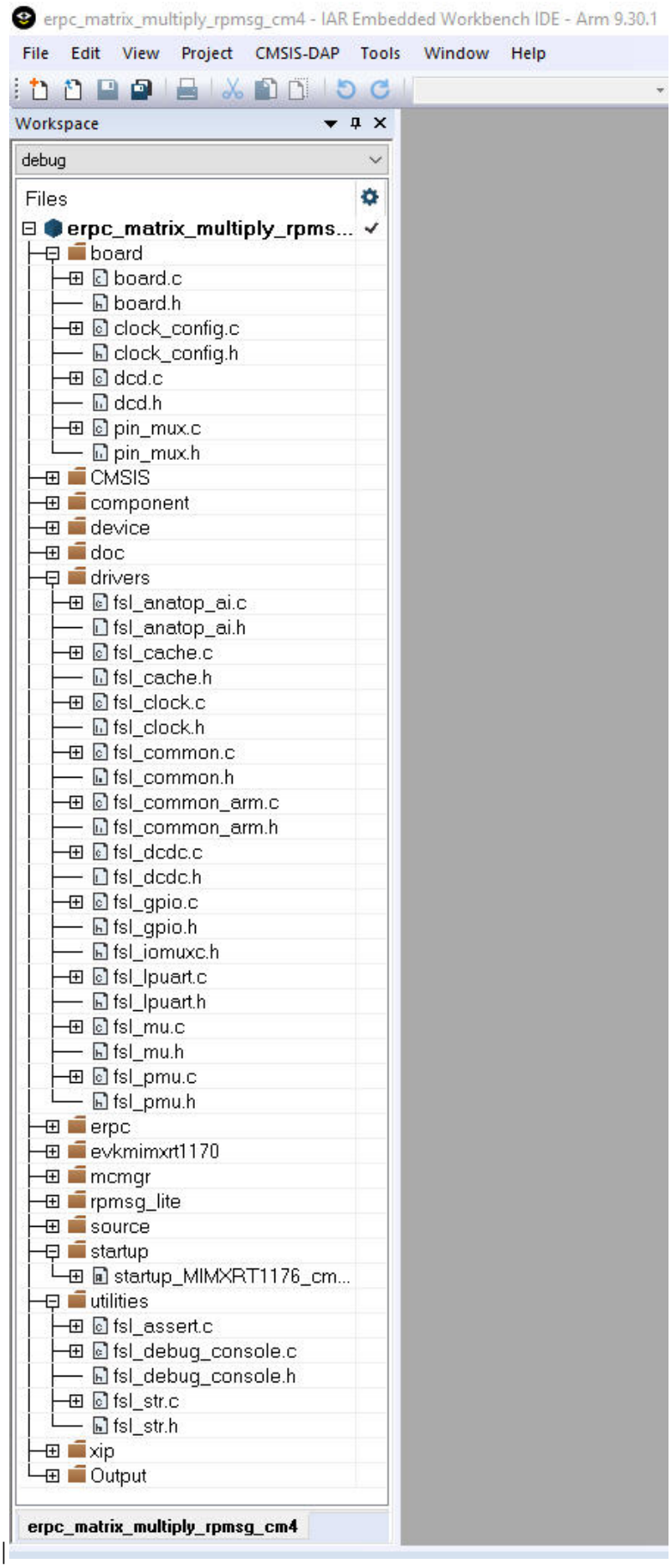
<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar

The project files for the eRPC server have the _cm4 suffix.

Server project basic source files

The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- *<MCUXpressoSDK_install_dir>/devices/<device>*
- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

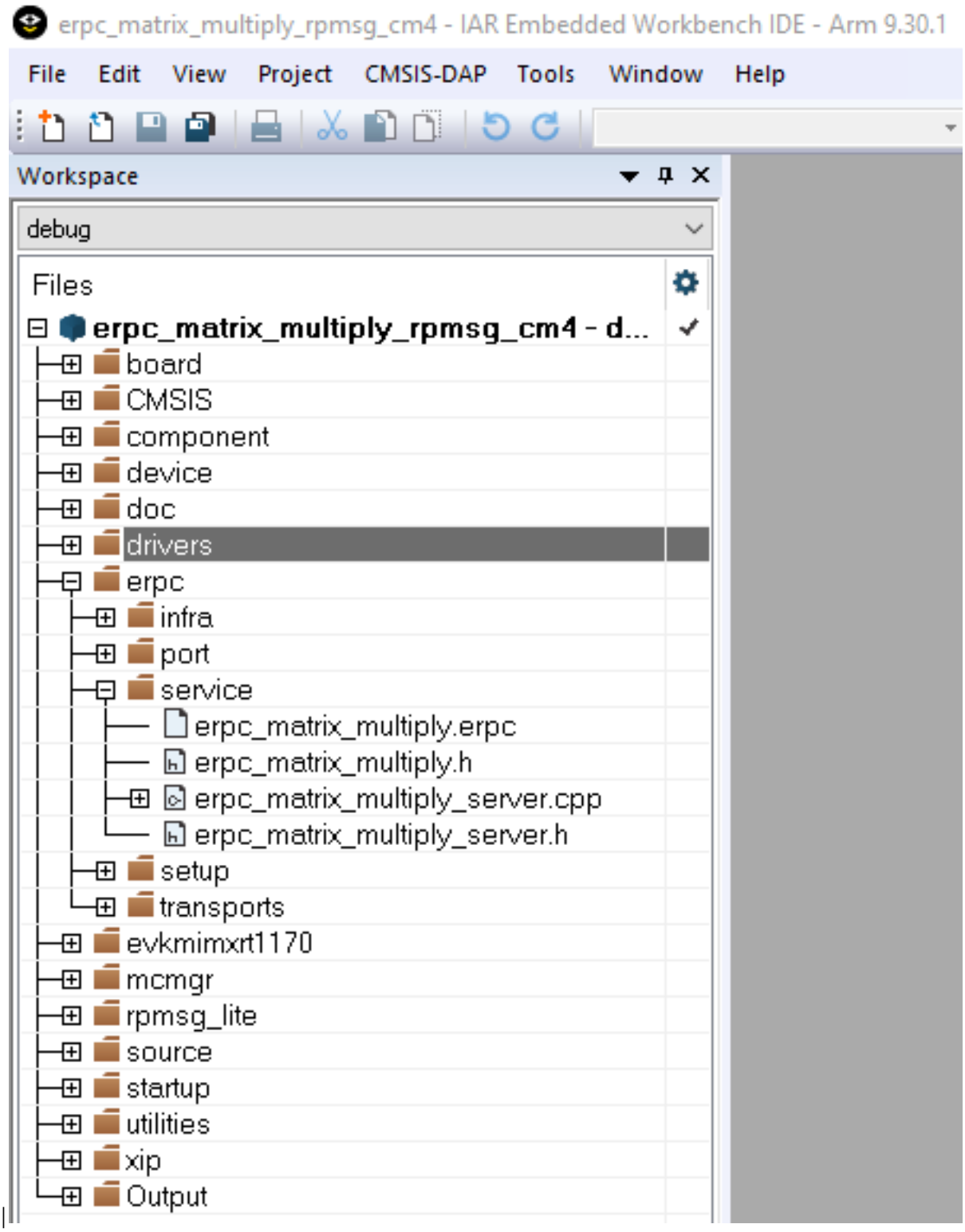
Server related generated files

The server-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/s



Parent topic: Multicore server application

Server infrastructure files

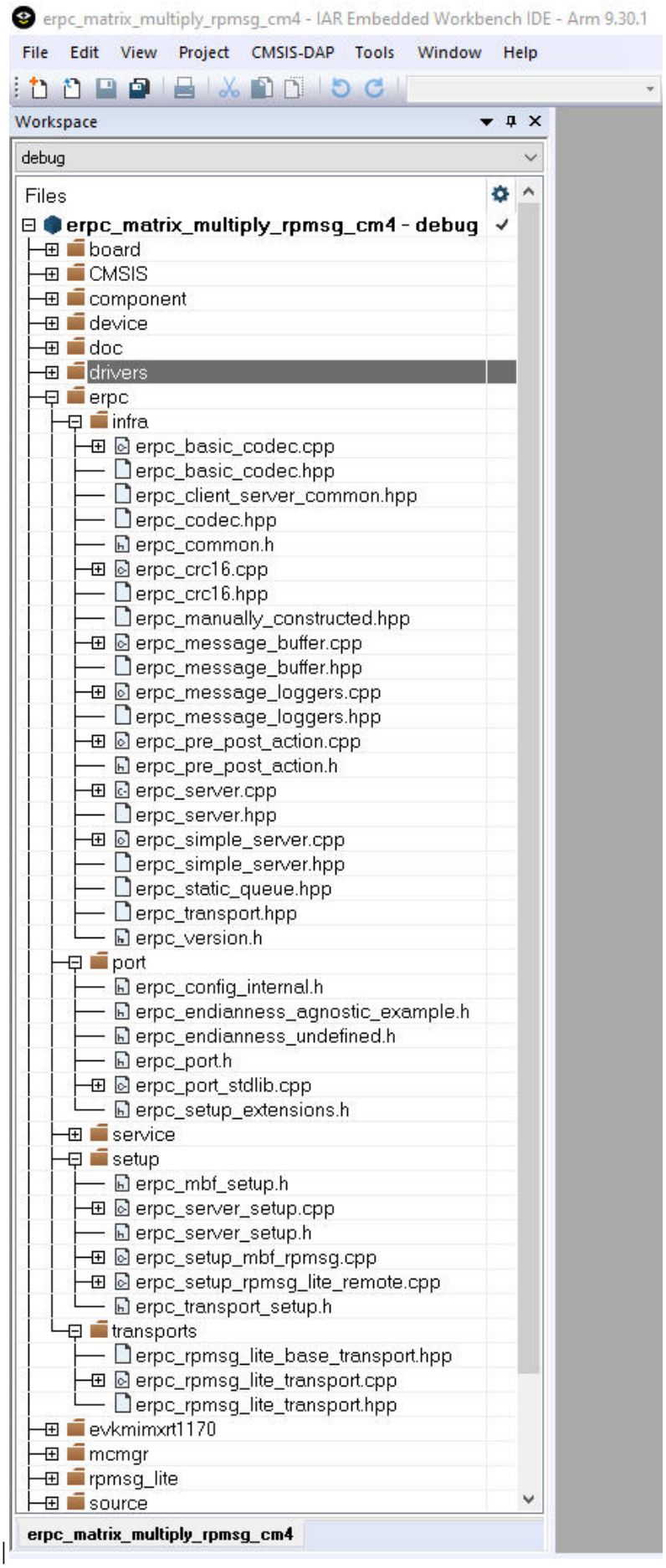
The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the *C/C++*

language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
  - Four files, erpc_server.hpp, erpc_server.cpp, erpc_simple_server.hpp, and erpc_simple_server.cpp, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
  - Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
  - The erpc_common.hpp file is used for common eRPC definitions, typedefs, and enums.
  - The erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
  - Message buffer files are used for storing serialized data: erpc_message_buffer.h and erpc_message_buffer.cpp.
  - The erpc_transport.h file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
  - erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
  - erpc_port_stdlib.cpp file ensures adaptation to stdlib.
  - erpc_config_internal.h internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
  - The erpc_server_setup.h and erpc_server_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
  - The erpc_transport_setup.h and erpc_setup_rpmsg_lite_remote.cpp files needs to be added into the project in order to allow the C-wrapped function for transport layer setup.
  - The erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
  - RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files need to be added into the server project.


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_server.cpp
erpc_server.hpp
erpc_simple_server.cpp
erpc_simple_server.hpp
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_mbf_setup.h
erpc_server_setup.cpp
erpc_server_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_remote.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

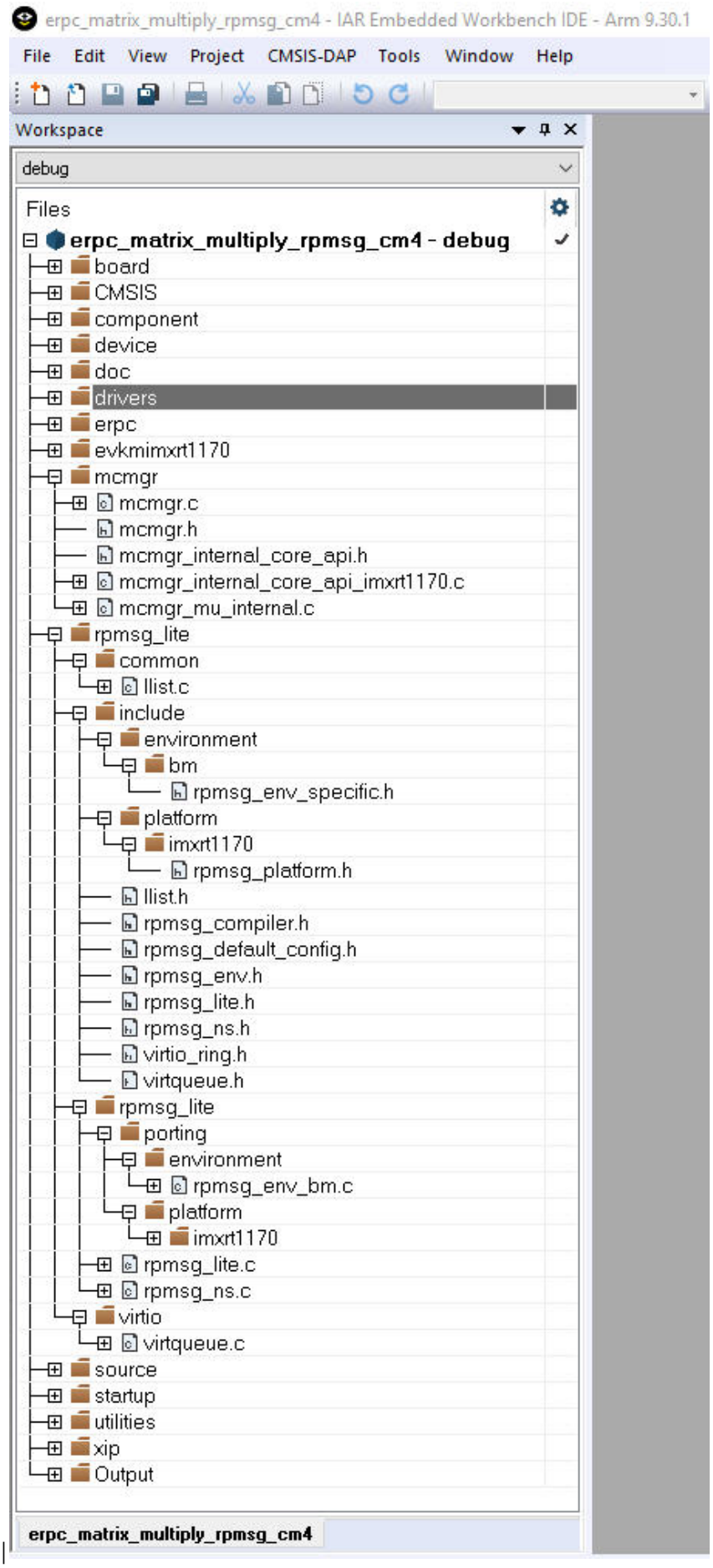
Server multicore infrastructure files

Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

Server user code

The server's user code is stored in the main_core1.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4

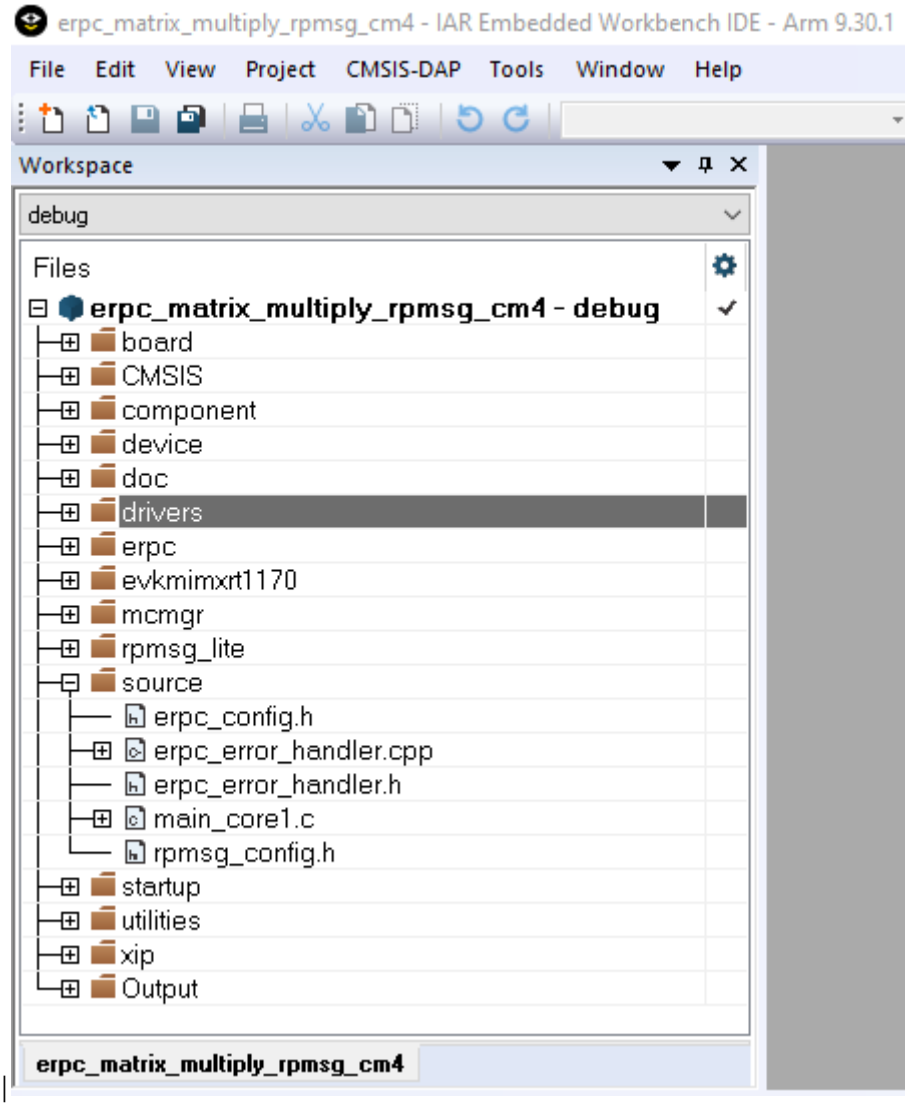
The main_core1.c file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the erpc_error_handler.h and erpc_error_handler.cpp files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
  ...
}
int main()
{
  ...
  /* RPMsg-Lite transport layer initialization */
  erpc_transport_t transport;
  transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
  ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_rpmsg_init(transport);
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server);
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg* folder.



Parent topic: Multicore server application

Parent topic:*Create an eRPC application*

Multicore client application

The “Matrix multiply” eRPC client project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar

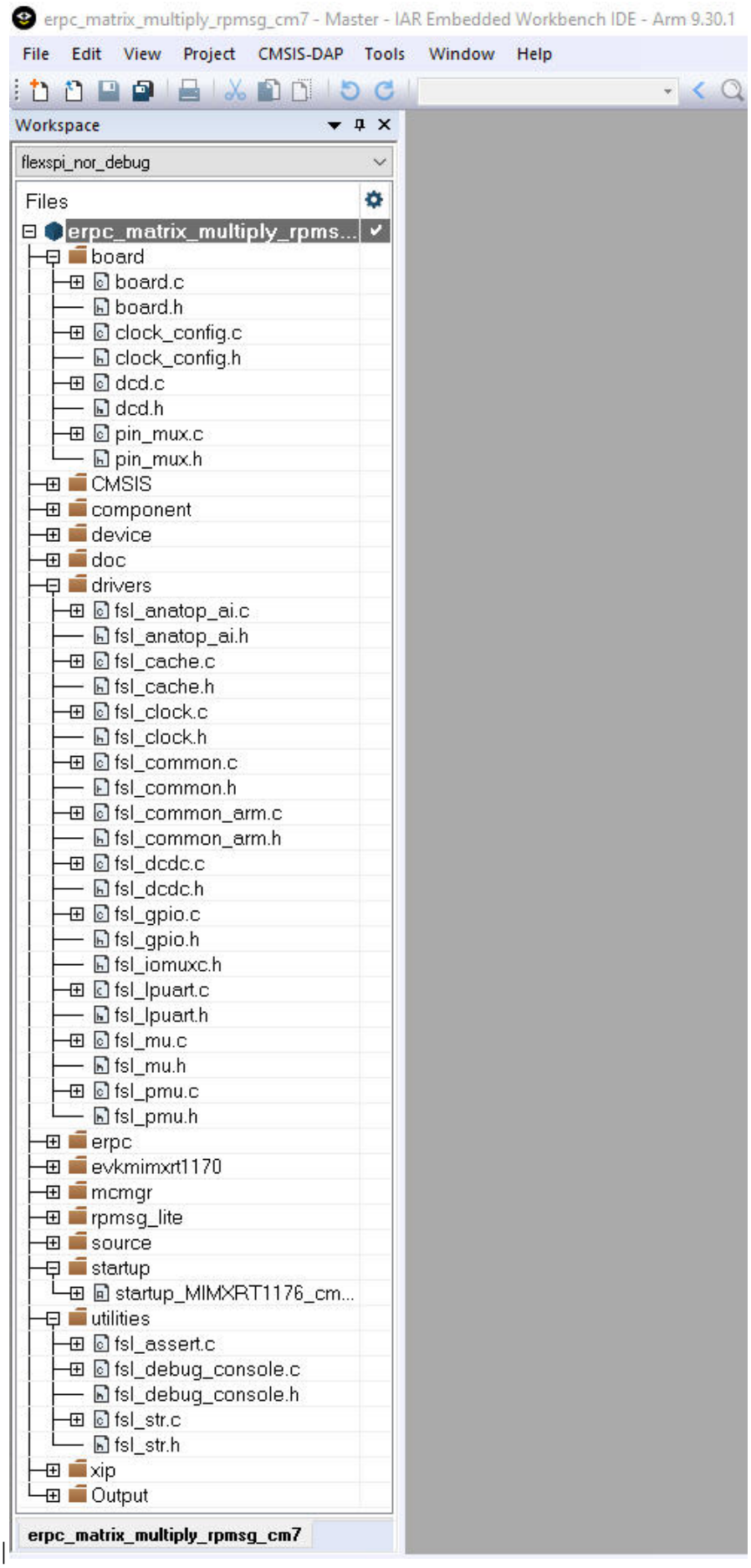
Project files for the eRPC client have the _cm7 suffix.

Client project basic source files

The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- *<MCUXpressoSDK_install_dir>/devices/<device>*

- *<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/*


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

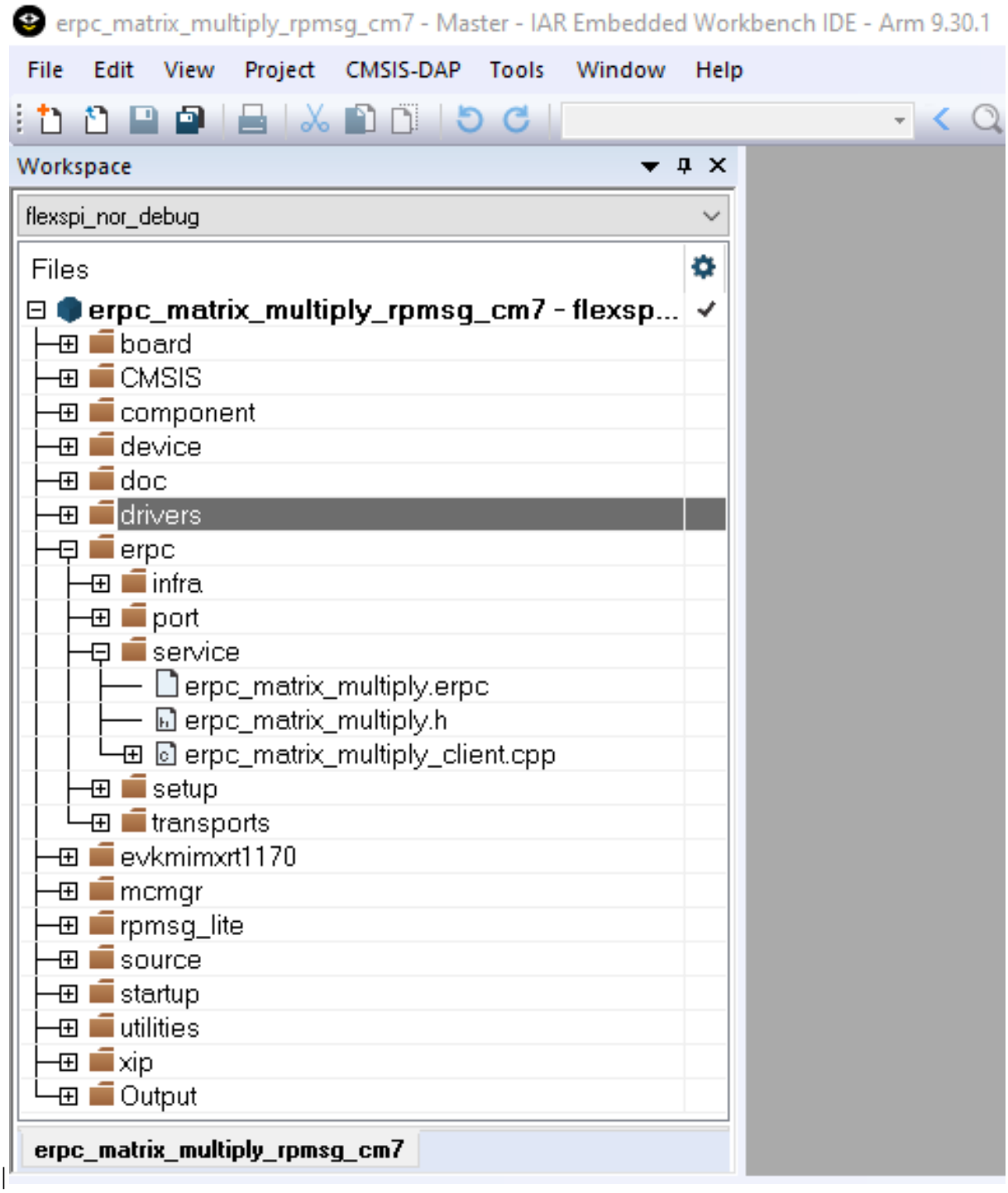
Parent topic: Multicore client application

Client-related generated files

The client-related generated files are:

- erpc_matric_multiply.h
- erpc_matrix_multiply_client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the *<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/* folder.



Parent topic: Multicore client application

Client infrastructure files

The eRPC infrastructure files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c

The **erpc_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, erpc_client_manager.h and erpc_client_manager.cpp, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (erpc_codec.hpp, erpc_basic_codec.hpp, and erpc_basic_codec.cpp) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- erpc_common.h file is used for common eRPC definitions, typedefs, and enums.
- erpc_manually_constructed.hpp file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: erpc_message_buffer.hpp and erpc_message_buffer.cpp.
- erpc_transport.hpp file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

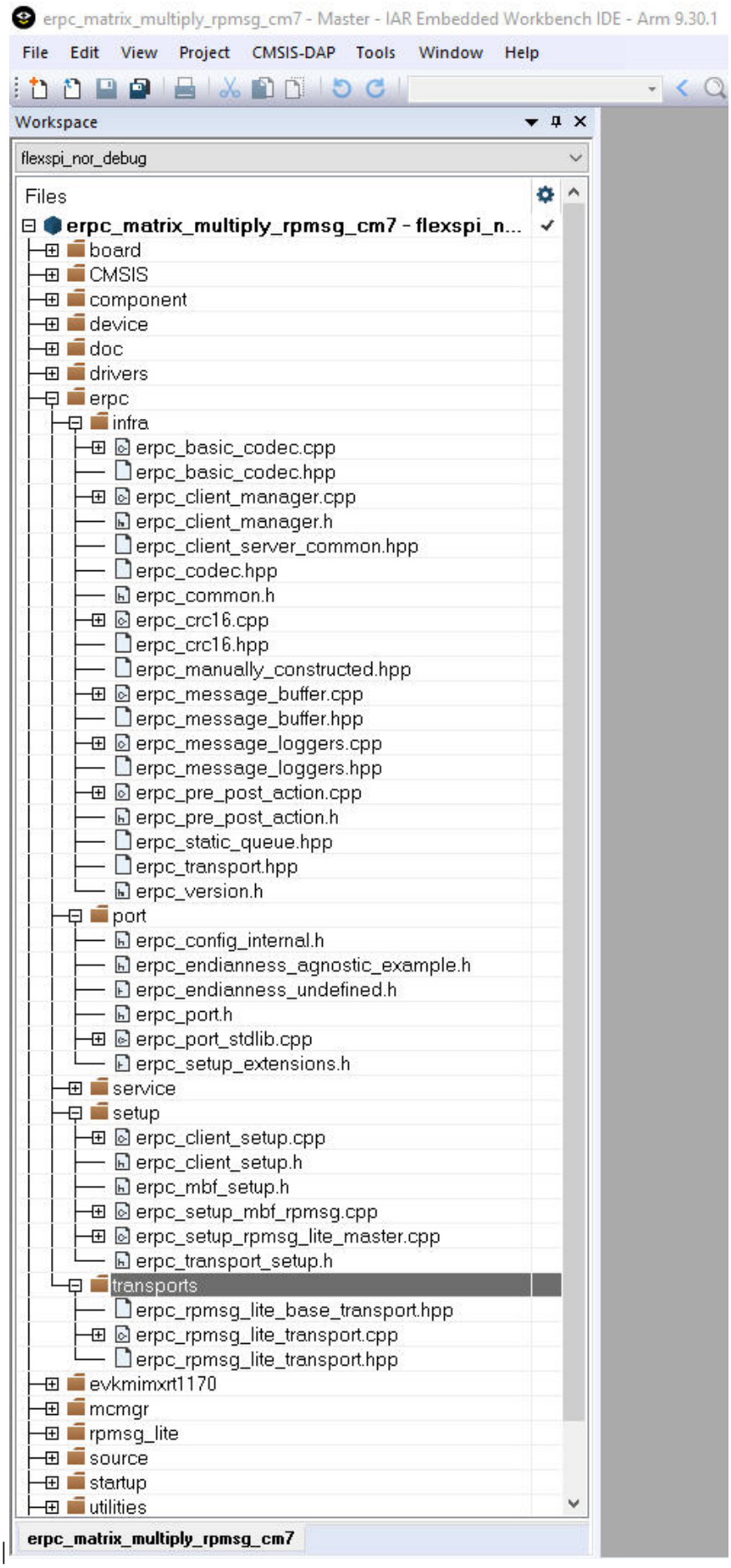
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- erpc_client_setup.h and erpc_client_setup.cpp files needs to be added into the "Matrix multiply" example project to demonstrate the use of C-wrapped functions in this example.
- erpc_transport_setup.h and erpc_setup_rpmsg_lite_master.cpp files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- erpc_mbf_setup.h and erpc_setup_mbf_rpmsg.cpp files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, erpc_rpmsg_lite_base_transport.hpp, erpc_rpmsg_lite_transport.hpp, and erpc_rpmsg_lite_transport.cpp files needs to be added into the client project.


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_n...
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_manager.cpp
erpc_client_manager.h
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_client_setup.cpp
erpc_client_setup.h
erpc_mbf_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_master.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
utilities
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

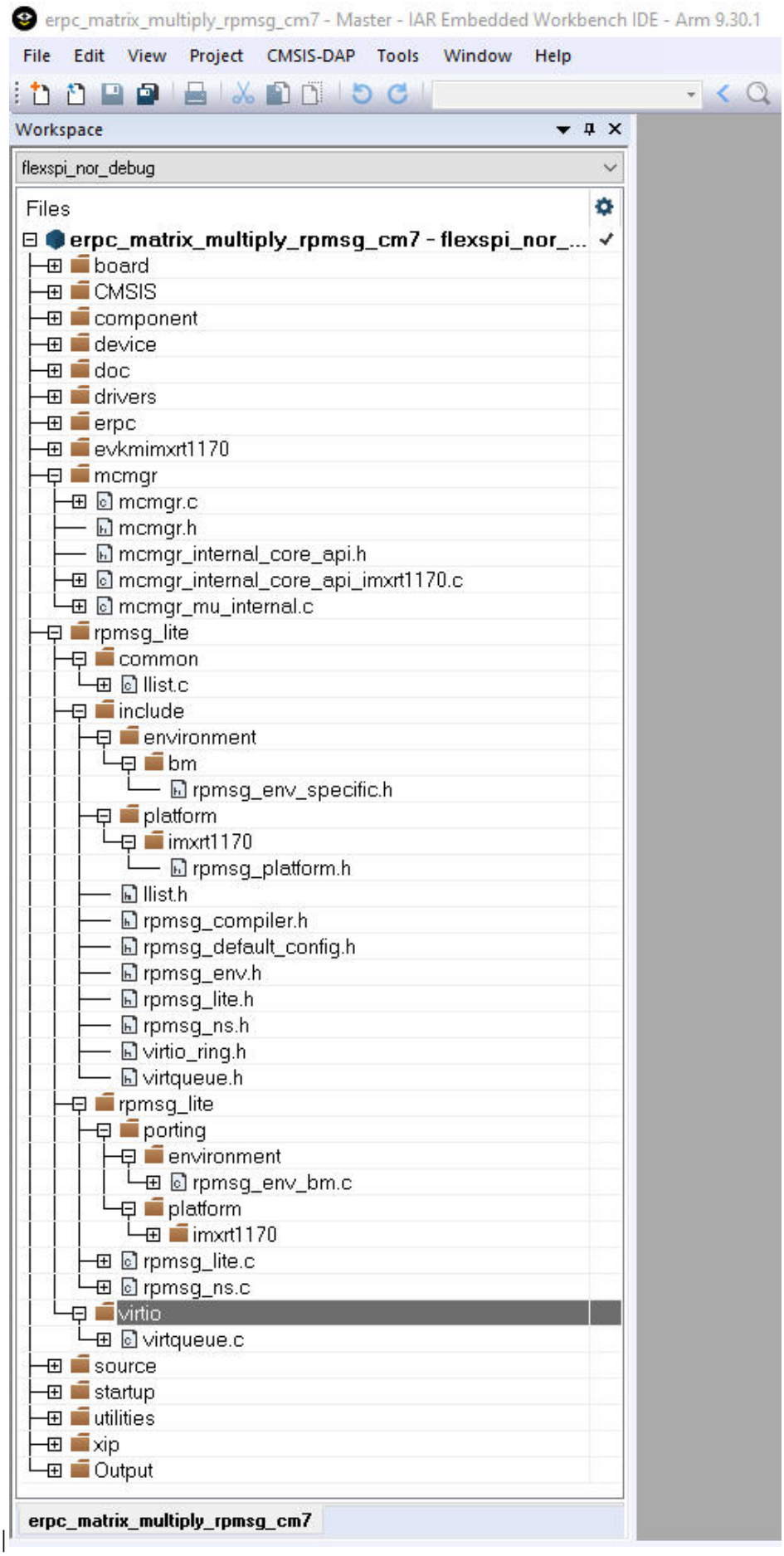
Client multicore infrastructure files

Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_nor_...
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

Parent topic: Multicore client application

Client user code

The client's user code is stored in the main_core0.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_example/erpc_matrix_multiply_rpmsg/cm7

The main_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc_error_handler.h and erpc_error_handler.cpp files.

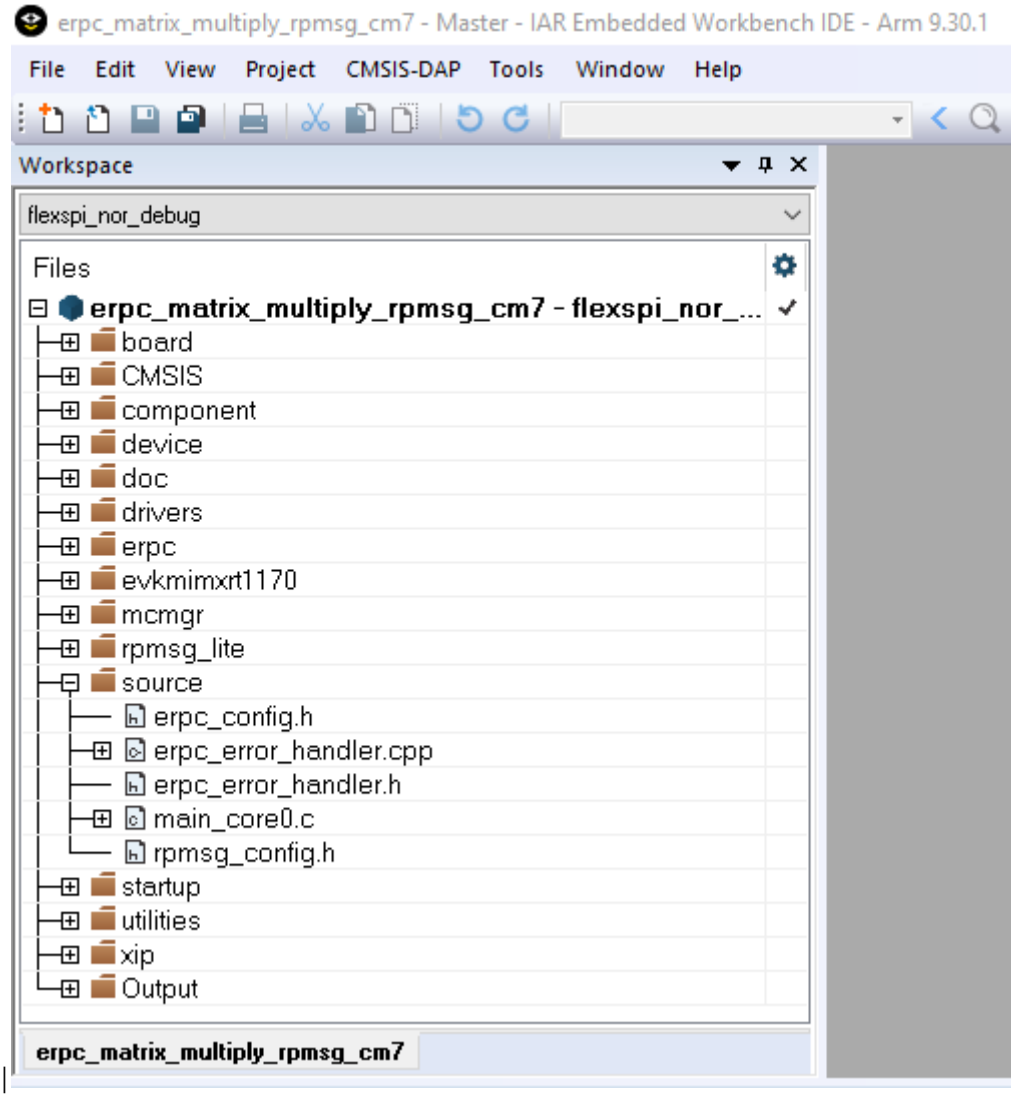
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg



Parent topic: Multicore client application

Parent topic:*Create an eRPC application*

Multiprocessor server application

The “Matrix multiply” eRPC server project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir»/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Server user code

The server's user code is stored in the main_server.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_lay* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
  ...
}
int main()
{
  ...
  /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
  erpc_transport_t transport;
  transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
  ...
  /* MessageBufferFactory initialization */
  erpc_mbf_t message_buffer_factory;
  message_buffer_factory = erpc_mbf_dynamic_init();
  ...
  /* eRPC server side initialization */
  erpc_server_t server;
  server = erpc_server_init(transport, message_buffer_factory);
  ...
  /* Adding the service to the server */
  erpc_service_t service = create_MatrixMultiplyService_service();
  erpc_add_service_to_server(server, service);
  ...
  while (1)
  {
    /* Process eRPC requests */
    erpc_status_t status = erpc_server_poll(server)
    /* handle error status */
    if (status != kErpcStatus_Success)
    {
      /* print error description */
      erpc_error_handler(status, 0);
      ...
    }
    ...
  }
}
```

Parent topic: Multiprocessor server application

Multiprocessor client application

The "Matrix multiply" eRPC client project for multiprocessor applications is located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<tr* folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RPMsg-Lite). The RPMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

|SPI|<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.hpp

<eRPC base directory>/erpc_c/transports/ erpc_(d)spi_master_transport.cpp

| |UART|<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/erpc_uart_cmsis_transport.cpp

|

Client user code

The client's user code is stored in the main_client.c file, located in the *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/* folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
  /* Invoke the erpcMatrixMultiply function */
  erpcMatrixMultiply(matrix1, matrix2, result_matrix);
  ...
  /* Check if some error occured in eRPC */
  if (g_erpc_error_occurred)
  {
    /* Exit program loop */
    break;
  }
  ...
}
```

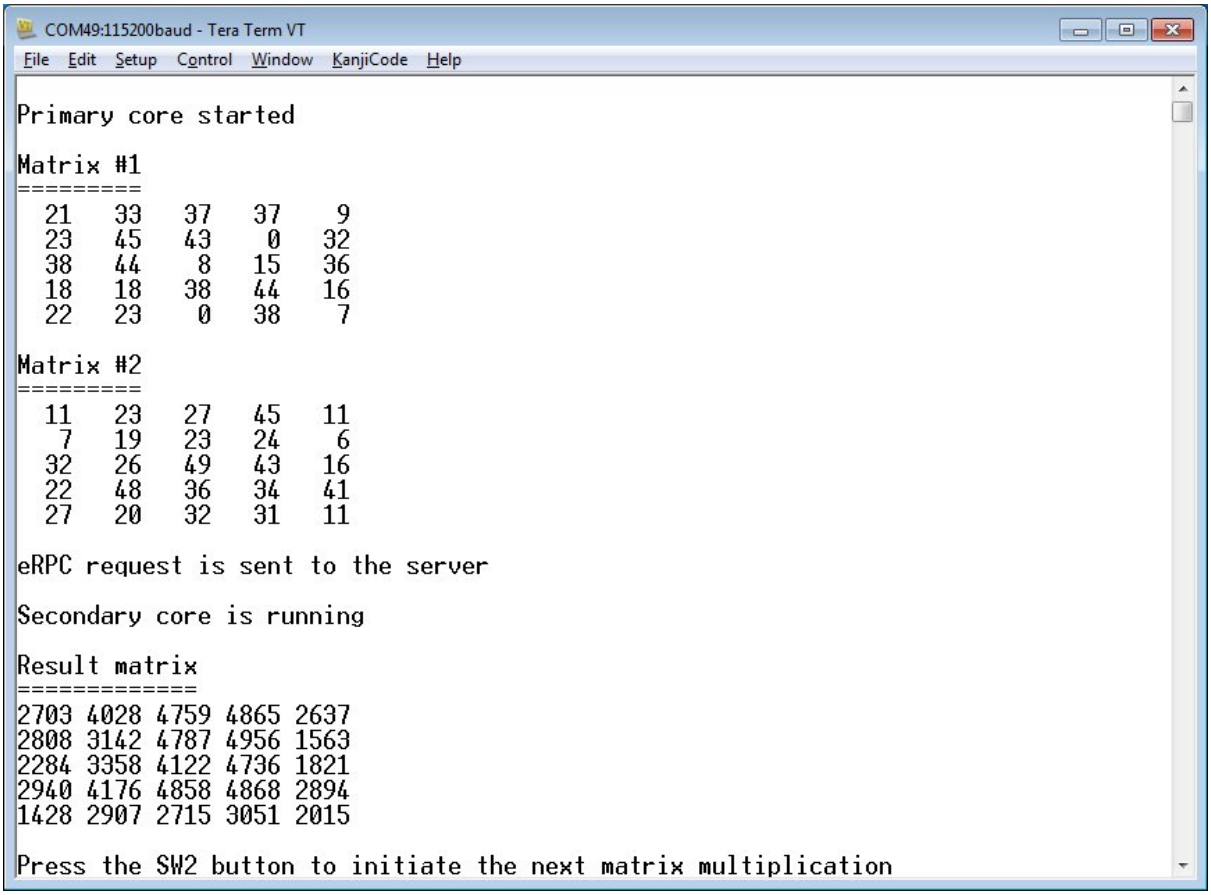
Parent topic: Multiprocessor client application

Parent topic: Multiprocessor server application

Parent topic:*Create an eRPC application*

Running the eRPC application

Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the <*MCUXpressoSDK_install_dir*>/*docs* folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:*Create an eRPC application*

Parent topic:*eRPC example*

Other uses for an eRPC implementation

The eRPC implementation is generic, and its use is not limited to just embedded applications. When creating an eRPC application outside the embedded world, the same principles apply. For example, this manual can be used to create an eRPC application for a PC running the Linux operating system. Based on the used type of transport medium, existing transport layers can be used, or new transport layers can be implemented.

For more information and erpc updates see the github.com/EmbeddedRPC.

Note about the source code in the document

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

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

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

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

Changelog eRPC

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

Unreleased

Added

- Added Zephyr IPC Service transport layer support.
- Zephyr examples added support for RT600 and RT700.

Updated

- eRPC Zephyr module port updated for Zephyr version 4.4

Fixed

- Python code of the eRPC infrastructure was updated to match the proper python code style, add type annotations and improve readability.
- eRPC: Several MISRA violations addressed.

1.14.0

Added

- Added Cmake/Kconfig support.
- Made java code jdk11 compliant, GitHub PR #432.
- Added imxrt1186 support into mu transport layer.
- erpcgen: Added assert for listType before usage, GitHub PR #406.

Fixed

- eRPC: Sources reformatted.
- erpc: Fixed typo in semaphore get (mutex -> semaphore), and write it can fail in case of timeout, GitHub PR #446.
- erpc: Free the arbitrated client token from client manager, GitHub PR #444.
- erpc: Fixed Makefile, install the erpc_simple_server header, GitHub PR #447.
- erpc_python: Fixed possible AttributeError and OSError on calling TCPTransport.close(), GitHub PR #438.
- Examples and tests consolidated.

1.13.0

Added

- erpc: Add BSD-3 license to endianness agnostic files, GitHub PR #417.
- eRPC: Add new Zephyr-related transports (zephyr_uart, zephyr_mbox).
- eRPC: Add new Zephyr-related examples.

Fixed

- eRPC,erpcgen: Fixing/improving markdown files, GitHub PR #395.
- eRPC: Fix Python client TCPTransports not being able to close, GitHub PR #390.
- eRPC,erpcgen: Align switch brackets, GitHub PR #396.
- erpc: Fix zephyr uart transport, GitHub PR #410.
- erpc: UART ZEPHYR Transport stop to work after a few transactions when using USB-CDC resolved, GitHub PR #420.

Removed

- eRPC,erpcgen: Remove cstbool library, GitHub PR #403.

1.12.0

Added

- eRPC: Add dynamic/static option for transport init, GitHub PR #361.
- eRPC,erpcgen: Winsock2 support, GitHub PR #365.
- eRPC,erpcgen: Feature/support multiple clients, GitHub PR #271.
- eRPC,erpcgen: Feature/buffer head - Framed transport header data stored in MessageBuffer, GitHub PR #378.
- eRPC,erpcgen: Add experimental Java support.

Fixed

- eRPC: Fix receive error value for spidev, GitHub PR #363.
- eRPC: UartTransport::init adaptation to changed driver.
- eRPC: Fix typo in assert, GitHub PR #371.
- eRPC,erpcgen: Move enums to enum classes, GitHub PR #379.
- eRPC: Fixed rpmsg tty transport to work with serial transport, GitHub PR #373.

1.11.0

Fixed

- eRPC: Makefiles update, GitHub PR #301.
- eRPC: Resolving warnings in Python, GitHub PR #325.
- eRPC: Python3.8 is not ready for usage of typing.Any type, GitHub PR #325.
- eRPC: Improved codec function to use reference instead of address, GitHub PR #324.
- eRPC: Fix NULL check for pending client creation, GitHub PR #341.
- eRPC: Replace sprintf with snprintf, GitHub PR #343.
- eRPC: Use MU_SendMsg blocking call in MU transport.
- eRPC: New LPSPI and LPI2C transport layers.
- eRPC: Freeing static objects, GitHub PR #353.
- eRPC: Fixed casting in deinit functions, GitHub PR #354.
- eRPC: Align LIBUSBSIO.GetNumPorts API use with libusbsio python module v. 2.1.11.
- erpcgen: Renamed temp variable to more generic one, GitHub PR #321.
- erpcgen: Add check that string read is not more than max length, GitHub PR #328.
- erpcgen: Move to g++ in pytest, GitHub PR #335.
- erpcgen: Use build=release for make, GitHub PR #334.
- erpcgen: Removed boost dependency, GitHub PR #346.
- erpcgen: Mingw support, GitHub PR #344.
- erpcgen: VS build update, GitHub PR #347.
- erpcgen: Modified name for common types macro scope, GitHub PR #337.
- erpcgen: Fixed memcpy for template, GitHub PR #352.
- eRPC,erpcgen: Change default build target to release + adding artefacts, GitHub PR #334.
- eRPC,erpcgen: Remove redundant includes, GitHub PR #338.
- eRPC,erpcgen: Many minor code improvements, GitHub PR #323.

1.10.0

Fixed

- eRPC: MU transport layer switched to blocking MU_SendMsg() API use.

1.10.0

Added

- eRPC: Add TCP_NODELAY option to python, GitHub PR #298.

Fixed

- eRPC: MUTransport adaptation to new supported SoCs.
- eRPC: Simplifying CI with installing dependencies using shell script, GitHub PR #267.
- eRPC: Using event for waiting for sock connection in TCP python server, formatting python code, C specific includes, GitHub PR #269.

- eRPC: Endianness agnostic update, GitHub PR #276.
- eRPC: Assertion added for functions which are returning status on freeing memory, GitHub PR #277.
- eRPC: Fixed closing arbitrator server in unit tests, GitHub PR #293.
- eRPC: Makefile updated to reflect the correct header names, GitHub PR #295.
- eRPC: Compare value length to used length() in reading data from message buffer, GitHub PR #297.
- eRPC: Replace EXPECT_TRUE with EXPECT_EQ in unit tests, GitHub PR #318.
- eRPC: Adapt rpmsg_lite based transports to changed rpmsg_lite_wait_for_link_up() API parameters.
- eRPC, erpcgen: Better distuingish which file can and cannot by linked by C linker, GitHub PR #266.
- eRPC, erpcgen: Stop checking if pointer is NULL before sending it to the erpc_free function, GitHub PR #275.
- eRPC, erpcgen: Changed api to count with more interfaces, GitHub PR #304.
- erpcgen: Check before reading from heap the buffer boundaries, GitHub PR #287.
- erpcgen: Several fixes for tests and CI, GitHub PR #289.
- erpcgen: Refactoring erpcgen code, GitHub PR #302.
- erpcgen: Fixed assigning const value to enum, GitHub PR #309.
- erpcgen: Enable runTesttest_enumErrorCode_allDirection, serialize enums as int32 instead of uint32.

1.9.1

Fixed

- eRPC: Construct the USB CDC transport, rather than a client, GitHub PR #220.
- eRPC: Fix premature import of package, causing failure when attempting installation of Python library in a clean environment, GitHub PR #38, #226.
- eRPC: Improve python detection in make, GitHub PR #225.
- eRPC: Fix several warnings with deprecated call in pytest, GitHub PR #227.
- eRPC: Fix freeing union members when only default need be freed, GitHub PR #228.
- eRPC: Fix making test under Linux, GitHub PR #229.
- eRPC: Assert costumizing, GitHub PR #148.
- eRPC: Fix corrupt clientList bug in TransportArbitrator, GitHub PR #199.
- eRPC: Fix build issue when invoking g++ with -Wno-error=free-nonheap-object, GitHub PR #233.
- eRPC: Fix inout cases, GitHub PR #237.
- eRPC: Remove ERPC_PRE_POST_ACTION dependency on return type, GitHub PR #238.
- eRPC: Adding NULL to ptr when codec function failed, fixing memcpy when fail is present during deserialization, GitHub PR #253.
- eRPC: MessageBuffer usage improvement, GitHub PR #258.
- eRPC: Get rid for serial and enum34 dependency (enum34 is in python3 since 3.4 (from 2014)), GitHub PR #247.

- eRPC: Several MISRA violations addressed.
- eRPC: Fix timeout for Freertos semaphore, GitHub PR #251.
- eRPC: Use of rpmsg_lite_wait_for_link_up() in rpmsg_lite based transports, GitHub PR #223.
- eRPC: Fix codec nullptr dereferencing, GitHub PR #264.
- erpcgen: Fix two syntax errors in erpcgen Python output related to non-encapsulated unions, improved test for union, GitHub PR #206, #224.
- erpcgen: Fix serialization of list/binary types, GitHub PR #240.
- erpcgen: Fix empty list parsing, GitHub PR #72.
- erpcgen: Fix templates for malloc errors, GitHub PR #110.
- erpcgen: Get rid of encapsulated union declarations in global scale, improve enum usage in unions, GitHub PR #249, #250.
- erpcgen: Fix compile error:UniqueIdChecker.cpp:156:104:'sort' was not declared, GitHub PR #265.

1.9.0

Added

- eRPC: Allow used LIBUSBSIO device index being specified from the Python command line argument.

Fixed

- eRPC: Improving template usage, GitHub PR #153.
- eRPC: run_clang_format.py cleanup, GitHub PR #177.
- eRPC: Build TCP transport setup code into liberpc, GitHub PR #179.
- eRPC: Fix multiple definitions of g_client error, GitHub PR #180.
- eRPC: Fix memset past end of buffer in erpc_setup_mbf_static.cpp, GitHub PR #184.
- eRPC: Fix deprecated error with newer pytest version, GitHub PR #203.
- eRPC, erpcgen: Static allocation support and usage of rpmsg static FreeRTOSs related APi, GitHub PR #168, #169.
- erpcgen: Remove redundant module imports in erpcgen, GitHub PR #196.

1.8.1

Added

- eRPC: New i2c_slave_transport trasnport introduced.

Fixed

- eRPC: Fix misra erpc c, GitHub PR #158.
- eRPC: Allow conditional compilation of message_loggers and pre_post_action.
- eRPC: (D)SPI slave transports updated to avoid busy loops in rtos environments.
- erpcgen: Re-implement EnumMember::hasValue(), GitHub PR #159.
- erpcgen: Fixing several misra issues in shim code, erpcgen and unit tests updated, GitHub PR #156.

- erpcgen: Fix bison file, GitHub PR #156.

1.8.0

Added

- eRPC: Support win32 thread, GitHub PR #108.
- eRPC: Add mbed support for malloc() and free(), GitHub PR #92.
- eRPC: Introduced pre and post callbacks for eRPC call, GitHub PR #131.
- eRPC: Introduced new USB CDC transport.
- eRPC: Introduced new Linux spidev-based transport.
- eRPC: Added formatting extension for VSC, GitHub PR #134.
- erpcgen: Introduce ustring type for unsigned char and force cast to char*, GitHub PR #125.

Fixed

- eRPC: Update makefile.
- eRPC: Fixed warnings and error with using MessageLoggers, GitHub PR #127.
- eRPC: Extend error msg for python server service handle function, GitHub PR #132.
- eRPC: Update CMSIS UART transport layer to avoid busy loops in rtos environments, introduce semaphores.
- eRPC: SPI transport update to allow usage without handshaking GPIO.
- eRPC: Native _WIN32 erpc serial transport and threading.
- eRPC: Arbitrator deadlock fix, TCP transport updated, TCP setup functions introduced, GitHub PR #121.
- eRPC: Update of matrix_multiply.py example: Add –serial and –baud argument, GitHub PR #137.
- eRPC: Update of .clang-format, GitHub PR #140.
- eRPC: Update of erpc_framed_transport.cpp: return error if received message has zero length, GitHub PR #141.
- eRPC, erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136, #139.
- eRPC, erpcgen: Core re-formatted using Clang version 10.
- erpcgen: Enable deallocation in server shim code when callback/function pointer used as out parameter in IDL.
- erpcgen: Removed '$' character from generated symbol name in '_$union' suffix, GitHub PR #103.
- erpcgen: Resolved mismatch between C++ and Python for callback index type, GitHub PR #111.
- erpcgen: Python generator improvements, GitHub PR #100, #118.
- erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136.

1.7.4

Added

- eRPC: Support MU transport unit testing.
- eRPC: Adding mbed os support.

Fixed

- eRPC: Unit test code updated to handle service add and remove operations.
- eRPC: Several MISRA issues in rpmsg-based transports addressed.
- eRPC: Fixed Linux/TCP acceptance tests in release target.
- eRPC: Minor documentation updates, code formatting.
- erpcgen: Whitespace removed from C common header template.

1.7.3

Fixed

- eRPC: Improved the test_callbacks logic to be more understandable and to allow requested callback execution on the server side.
- eRPC: TransportArbitrator::prepareClientReceive modified to avoid incorrect return value type.
- eRPC: The ClientManager and the ArbitratedClientManager updated to avoid performing client requests when the previous serialization phase fails.
- erpcgen: Generate the shim code for destroy of statically allocated services.

1.7.2

Added

- eRPC: Add missing doxygen comments for transports.

Fixed

- eRPC: Improved support of const types.
- eRPC: Fixed Mac build.
- eRPC: Fixed serializing python list.
- eRPC: Documentation update.

1.7.1

Fixed

- eRPC: Fixed semaphore in static message buffer factory.
- erpcgen: Fixed MU received error flag.
- erpcgen: Fixed tcp transport.

1.7.0

Added

- eRPC: List names are based on their types. Names are more deterministic.
- eRPC: Service objects are as a default created as global static objects.

- eRPC: Added missing doxygen comments.
- eRPC: Added support for 64bit numbers.
- eRPC: Added support of program language specific annotations.

Fixed

- eRPC: Improved code size of generated code.
- eRPC: Generating crc value is optional.
- eRPC: Fixed CMSIS Uart driver. Removed dependency on KSDK.
- eRPC: Forbid users use reserved words.
- eRPC: Removed outByref for function parameters.
- eRPC: Optimized code style of callback functions.

1.6.0

Added

- eRPC: Added @nullable support for scalar types.

Fixed

- eRPC: Improved code size of generated code.
- eRPC: Improved eRPC nested calls.
- eRPC: Improved eRPC list length variable serialization.

1.5.0

Added

- eRPC: Added support for unions type non-wrapped by structure.
- eRPC: Added callbacks support.
- eRPC: Added support @external annotation for functions.
- eRPC: Added support @name annotation.
- eRPC: Added Messaging Unit transport layer.
- eRPC: Added RPMSG Lite RTOS TTY transport layer.
- eRPC: Added version verification and IDL version verification between eRPC code and eRPC generated shim code.
- eRPC: Added support of shared memory pointer.
- eRPC: Added annotation to forbid generating const keyword for function parameters.
- eRPC: Added python matrix multiply example.
- eRPC: Added nested call support.
- eRPC: Added struct member “byref” option support.
- eRPC: Added support of forward declarations of structures
- eRPC: Added Python RPMsg Multiendpoint kernel module support
- eRPC: Added eRPC sniffer tool

1.4.0

Added

- eRPC: New RPMsg-Lite Zero Copy (RPMsgZC) transport layer.

Fixed

- eRPC: win_flex_bison.zip for windows updated.
- eRPC: Use one codec (instead of inCodec outCodec).

[1.3.0]

Added

- eRPC: New annotation types introduced (@length, @max_length, ...).
- eRPC: Support for running both erpc client and erpc server on one side.
- eRPC: New transport layers for (LP)UART, (D)SPI.
- eRPC: Error handling support.

[1.2.0]

Added

- eRPC source directory organization changed.
- Many eRPC improvements.

[1.1.0]

Added

- Multicore SDK 1.1.0 ported to KSDK 2.0.0.

[1.0.0]

Added

- Initial Release

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.

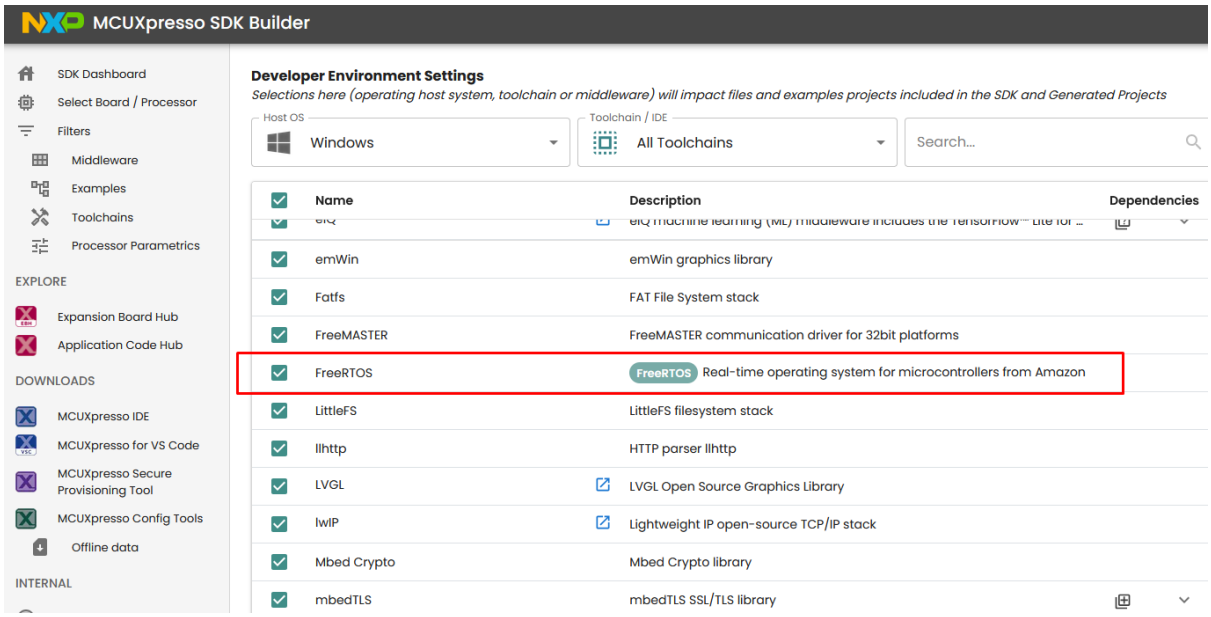
List of examples

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

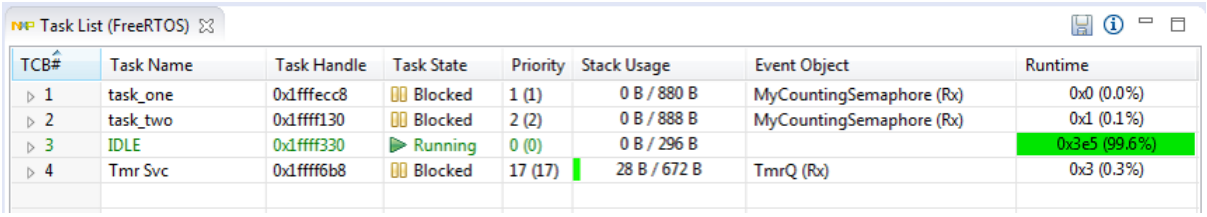
For information how to use the cmake and Kconfig based build and configuration system and how to build freertos_examples visit: MCUXpresso SDK documentation for Build And Configuration MCUXpresso SDK Getting Start Guide

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-frdmk22f.pdf — from *boards/Kinetis/frdmk22f/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_Kinetis_K_MK22F51212._adc16_channel_config (*C struct*), 148
drivers_Kinetis_K_MK22F51212._adc16_channel_config.channelNumber (*C var*), 148
drivers_Kinetis_K_MK22F51212._adc16_channel_config.enableDifferentialConversion (*C var*), 148
drivers_Kinetis_K_MK22F51212._adc16_channel_config.enableInterruptOnConversionCompleted (*C var*), 148
drivers_Kinetis_K_MK22F51212._adc16_channel_status_flags (*C enum*), 142
drivers_Kinetis_K_MK22F51212._adc16_channel_status_flags.kADC16_ChannelConversionDoneFlag (*C enumerator*), 142
drivers_Kinetis_K_MK22F51212._adc16_clock_divider (*C enum*), 143
drivers_Kinetis_K_MK22F51212._adc16_clock_divider.kADC16_ClockDivider1 (*C enumerator*), 143
drivers_Kinetis_K_MK22F51212._adc16_clock_divider.kADC16_ClockDivider2 (*C enumerator*), 143
drivers_Kinetis_K_MK22F51212._adc16_clock_divider.kADC16_ClockDivider4 (*C enumerator*), 143
drivers_Kinetis_K_MK22F51212._adc16_clock_divider.kADC16_ClockDivider8 (*C enumerator*), 143
drivers_Kinetis_K_MK22F51212._adc16_clock_source (*C enum*), 144
drivers_Kinetis_K_MK22F51212._adc16_clock_source.kADC16_ClockSourceAlt0 (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_clock_source.kADC16_ClockSourceAlt1 (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_clock_source.kADC16_ClockSourceAlt2 (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_clock_source.kADC16_ClockSourceAlt3 (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_clock_source.kADC16_ClockSourceAsynchronousClock (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_config (*C struct*), 147
drivers_Kinetis_K_MK22F51212._adc16_config.clockDivider (*C var*), 147
drivers_Kinetis_K_MK22F51212._adc16_config.clockSource (*C var*), 147
drivers_Kinetis_K_MK22F51212._adc16_config.enableAsynchronousClock (*C var*), 147
drivers_Kinetis_K_MK22F51212._adc16_config.enableContinuousConversion (*C var*), 148
drivers_Kinetis_K_MK22F51212._adc16_config.enableHighSpeed (*C var*), 148
drivers_Kinetis_K_MK22F51212._adc16_config.enableLowPower (*C var*), 148
drivers_Kinetis_K_MK22F51212._adc16_config.hardwareAverageMode (*C var*), 148
drivers_Kinetis_K_MK22F51212._adc16_config.longSampleMode (*C var*), 147
drivers_Kinetis_K_MK22F51212._adc16_config.referenceVoltageSource (*C var*), 147
drivers_Kinetis_K_MK22F51212._adc16_config.resolution (*C var*), 147
drivers_Kinetis_K_MK22F51212._adc16_hardware_average_mode (*C enum*), 145
drivers_Kinetis_K_MK22F51212._adc16_hardware_average_mode.kADC16_HardwareAverageCount4 (*C enumerator*), 145
drivers_Kinetis_K_MK22F51212._adc16_hardware_average_mode.kADC16_HardwareAverageCount8 (*C enumerator*), 145
drivers_Kinetis_K_MK22F51212._adc16_hardware_average_mode.kADC16_HardwareAverageCount16

(*C enumerator*), 145
drivers_Kinetis_K_MK22F51212._adc16_hardware_average_mode.kADC16_HardwareAverageCount32 (*C enumerator*), 145
drivers_Kinetis_K_MK22F51212._adc16_hardware_average_mode.kADC16_HardwareAverageDisabled (*C enumerator*), 145
drivers_Kinetis_K_MK22F51212._adc16_hardware_compare_config (*C struct*), 148
drivers_Kinetis_K_MK22F51212._adc16_hardware_compare_config.hardwareCompareMode (*C var*), 148
drivers_Kinetis_K_MK22F51212._adc16_hardware_compare_config.value1 (*C var*), 148
drivers_Kinetis_K_MK22F51212._adc16_hardware_compare_config.value2 (*C var*), 148
drivers_Kinetis_K_MK22F51212._adc16_hardware_compare_mode (*C enum*), 145
drivers_Kinetis_K_MK22F51212._adc16_hardware_compare_mode.kADC16_HardwareCompareMode0 (*C enumerator*), 145
drivers_Kinetis_K_MK22F51212._adc16_hardware_compare_mode.kADC16_HardwareCompareMode1 (*C enumerator*), 145
drivers_Kinetis_K_MK22F51212._adc16_hardware_compare_mode.kADC16_HardwareCompareMode2 (*C enumerator*), 146
drivers_Kinetis_K_MK22F51212._adc16_hardware_compare_mode.kADC16_HardwareCompareMode3 (*C enumerator*), 146
drivers_Kinetis_K_MK22F51212._adc16_long_sample_mode (*C enum*), 144
drivers_Kinetis_K_MK22F51212._adc16_long_sample_mode.kADC16_LongSampleCycle6 (*C enumerator*), 145
drivers_Kinetis_K_MK22F51212._adc16_long_sample_mode.kADC16_LongSampleCycle10 (*C enumerator*), 145
drivers_Kinetis_K_MK22F51212._adc16_long_sample_mode.kADC16_LongSampleCycle16 (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_long_sample_mode.kADC16_LongSampleCycle24 (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_long_sample_mode.kADC16_LongSampleDisabled (*C enumerator*), 145
drivers_Kinetis_K_MK22F51212._adc16_pga_config (*C struct*), 148
drivers_Kinetis_K_MK22F51212._adc16_pga_config.disablePgaChopping (*C var*), 149
drivers_Kinetis_K_MK22F51212._adc16_pga_config.enableRunInNormalMode (*C var*), 149
drivers_Kinetis_K_MK22F51212._adc16_pga_config.enableRunInOffsetMeasurement (*C var*), 149
drivers_Kinetis_K_MK22F51212._adc16_pga_config.pgaGain (*C var*), 149
drivers_Kinetis_K_MK22F51212._adc16_pga_gain (*C enum*), 146
drivers_Kinetis_K_MK22F51212._adc16_pga_gain.kADC16_PGAGainValueOf1 (*C enumerator*), 146
drivers_Kinetis_K_MK22F51212._adc16_pga_gain.kADC16_PGAGainValueOf2 (*C enumerator*), 146
drivers_Kinetis_K_MK22F51212._adc16_pga_gain.kADC16_PGAGainValueOf4 (*C enumerator*), 146
drivers_Kinetis_K_MK22F51212._adc16_pga_gain.kADC16_PGAGainValueOf8 (*C enumerator*), 146
drivers_Kinetis_K_MK22F51212._adc16_pga_gain.kADC16_PGAGainValueOf16 (*C enumerator*), 146
drivers_Kinetis_K_MK22F51212._adc16_pga_gain.kADC16_PGAGainValueOf32 (*C enumerator*), 146
drivers_Kinetis_K_MK22F51212._adc16_pga_gain.kADC16_PGAGainValueOf64 (*C enumerator*), 146
drivers_Kinetis_K_MK22F51212._adc16_reference_voltage_source (*C enum*), 145
drivers_Kinetis_K_MK22F51212._adc16_reference_voltage_source.kADC16_ReferenceVoltageSourceValt (*C enumerator*), 145
drivers_Kinetis_K_MK22F51212._adc16_reference_voltage_source.kADC16_ReferenceVoltageSourceVref (*C enumerator*), 145
drivers_Kinetis_K_MK22F51212._adc16_resolution (*C enum*), 143
drivers_Kinetis_K_MK22F51212._adc16_resolution.kADC16_Resolution8or9Bit (*C enumerator*), 143

drivers_Kinetis_K_MK22F51212._adc16_resolution.kADC16_Resolution10or11Bit (*C enumerator*), 143
drivers_Kinetis_K_MK22F51212._adc16_resolution.kADC16_Resolution12or13Bit (*C enumerator*), 143
drivers_Kinetis_K_MK22F51212._adc16_resolution.kADC16_ResolutionDF9Bit (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_resolution.kADC16_ResolutionDF11Bit (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_resolution.kADC16_ResolutionDF13Bit (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_resolution.kADC16_ResolutionSE8Bit (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_resolution.kADC16_ResolutionSE10Bit (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_resolution.kADC16_ResolutionSE12Bit (*C enumerator*), 144
drivers_Kinetis_K_MK22F51212._adc16_status_flags (*C enum*), 142
drivers_Kinetis_K_MK22F51212._adc16_status_flags.kADC16_ActiveFlag (*C enumerator*), 143
drivers_Kinetis_K_MK22F51212._adc16_status_flags.kADC16_CalibrationFailedFlag (*C enumerator*), 143
drivers_Kinetis_K_MK22F51212._adc_channel_mux_mode (*C enum*), 143
drivers_Kinetis_K_MK22F51212._adc_channel_mux_mode.kADC16_ChannelMuxA (*C enumerator*), 143
drivers_Kinetis_K_MK22F51212._adc_channel_mux_mode.kADC16_ChannelMuxB (*C enumerator*), 143
drivers_Kinetis_K_MK22F51212._clock_ip_name (*C enum*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Adc0 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Adc1 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Cmp0 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Cmp1 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Crc0 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Dac0 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Dac1 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Dma0 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Dmamux0 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Ewm0 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Flexbus0 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Ftf0 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Ftm0 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Ftm1 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Ftm2 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Ftm3 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_I2c0 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_I2c1 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_IpInvalid (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Lptmr0 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Lpuart0 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Pdb0 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Pit0 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_PortA (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_PortB (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_PortC (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_PortD (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_PortE (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Rnga0 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Rtc0 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Sai0 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Spi0 (*C enumerator*), 152

drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Spi1 (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Uart0 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Uart1 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Uart2 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Usbfs0 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_ip_name.kCLOCK_Vref0 (*C enumerator*), 151
drivers_Kinetis_K_MK22F51212._clock_name (*C enum*), 149
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_BusClk (*C enumerator*), 149
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_CoreSysClk (*C enumerator*), 149
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_Er32kClk (*C enumerator*), 149
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_FastPeriphClk (*C enumerator*), 149
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_FlashClk (*C enumerator*), 149
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_FlexBusClk (*C enumerator*), 149
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_LpoClk (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_McgExtPllClk (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_McgFixedFreqClk (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_McgFllClk (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_McgInternalRefClk (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_McgIrc48MClk (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_McgPeriphClk (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_McgPll0Clk (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_McgPll1Clk (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_Osc0ErClk (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_Osc0ErClkUndiv (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_Osc1ErClk (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_PlatClk (*C enumerator*), 149
drivers_Kinetis_K_MK22F51212._clock_name.kCLOCK_PllFllSelClk (*C enumerator*), 149
drivers_Kinetis_K_MK22F51212._clock_usb_src (*C enum*), 150
drivers_Kinetis_K_MK22F51212._clock_usb_src.kCLOCK_UsbSrcExt (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_usb_src.kCLOCK_UsbSrcIrc48M (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._clock_usb_src.kCLOCK_UsbSrcPll0 (*C enumerator*), 150
drivers_Kinetis_K_MK22F51212._cmp_config (*C struct*), 181
drivers_Kinetis_K_MK22F51212._cmp_config.enableCmp (*C var*), 181
drivers_Kinetis_K_MK22F51212._cmp_config.enableHighSpeed (*C var*), 181
drivers_Kinetis_K_MK22F51212._cmp_config.enableInvertOutput (*C var*), 181
drivers_Kinetis_K_MK22F51212._cmp_config.enablePinOut (*C var*), 181
drivers_Kinetis_K_MK22F51212._cmp_config.enableTriggerMode (*C var*), 181
drivers_Kinetis_K_MK22F51212._cmp_config.hysteresisMode (*C var*), 181
drivers_Kinetis_K_MK22F51212._cmp_config.useUnfilteredOutput (*C var*), 181
drivers_Kinetis_K_MK22F51212._cmp_dac_config (*C struct*), 181
drivers_Kinetis_K_MK22F51212._cmp_dac_config.DACValue (*C var*), 182
drivers_Kinetis_K_MK22F51212._cmp_dac_config.referenceVoltageSource (*C var*), 182
drivers_Kinetis_K_MK22F51212._cmp_filter_config (*C struct*), 181
drivers_Kinetis_K_MK22F51212._cmp_filter_config.enableSample (*C var*), 181
drivers_Kinetis_K_MK22F51212._cmp_filter_config.filterCount (*C var*), 181
drivers_Kinetis_K_MK22F51212._cmp_filter_config.filterPeriod (*C var*), 181
drivers_Kinetis_K_MK22F51212._cmp_hysteresis_mode (*C enum*), 180
drivers_Kinetis_K_MK22F51212._cmp_hysteresis_mode.kCMP_HysteresisLevel0 (*C enumerator*), 180
drivers_Kinetis_K_MK22F51212._cmp_hysteresis_mode.kCMP_HysteresisLevel1 (*C enumerator*), 180
drivers_Kinetis_K_MK22F51212._cmp_hysteresis_mode.kCMP_HysteresisLevel2 (*C enumerator*), 180
drivers_Kinetis_K_MK22F51212._cmp_hysteresis_mode.kCMP_HysteresisLevel3 (*C enumerator*), 180
drivers_Kinetis_K_MK22F51212._cmp_interrupt_enable (*C enum*), 179
drivers_Kinetis_K_MK22F51212._cmp_interrupt_enable.kCMP_OutputFallingInterruptEnable (*C enumerator*), 179

drivers_Kinetis_K_MK22F51212._cmp_interrupt_enable.kCMP_OutputRisingInterruptEnable (*C enumerator*), 179
drivers_Kinetis_K_MK22F51212._cmp_reference_voltage_source (*C enum*), 180
drivers_Kinetis_K_MK22F51212._cmp_reference_voltage_source.kCMP_VrefSourceVin1 (*C enumerator*), 180
drivers_Kinetis_K_MK22F51212._cmp_reference_voltage_source.kCMP_VrefSourceVin2 (*C enumerator*), 180
drivers_Kinetis_K_MK22F51212._cmp_status_flags (*C enum*), 179
drivers_Kinetis_K_MK22F51212._cmp_status_flags.kCMP_OutputAssertEventFlag (*C enumerator*), 180
drivers_Kinetis_K_MK22F51212._cmp_status_flags.kCMP_OutputFallingEventFlag (*C enumerator*), 180
drivers_Kinetis_K_MK22F51212._cmp_status_flags.kCMP_OutputRisingEventFlag (*C enumerator*), 179
drivers_Kinetis_K_MK22F51212._crc_bits (*C enum*), 182
drivers_Kinetis_K_MK22F51212._crc_bits.kCrcBits16 (*C enumerator*), 182
drivers_Kinetis_K_MK22F51212._crc_bits.kCrcBits32 (*C enumerator*), 182
drivers_Kinetis_K_MK22F51212._crc_config (*C struct*), 184
drivers_Kinetis_K_MK22F51212._crc_config.complementChecksum (*C var*), 185
drivers_Kinetis_K_MK22F51212._crc_config.crcBits (*C var*), 185
drivers_Kinetis_K_MK22F51212._crc_config.crcResult (*C var*), 185
drivers_Kinetis_K_MK22F51212._crc_config.polynomial (*C var*), 184
drivers_Kinetis_K_MK22F51212._crc_config.reflectIn (*C var*), 184
drivers_Kinetis_K_MK22F51212._crc_config.reflectOut (*C var*), 185
drivers_Kinetis_K_MK22F51212._crc_config.seed (*C var*), 184
drivers_Kinetis_K_MK22F51212._crc_result (*C enum*), 182
drivers_Kinetis_K_MK22F51212._crc_result.kCrcFinalChecksum (*C enumerator*), 182
drivers_Kinetis_K_MK22F51212._crc_result.kCrcIntermediateChecksum (*C enumerator*), 183
drivers_Kinetis_K_MK22F51212._dac_buffer_config (*C struct*), 190
drivers_Kinetis_K_MK22F51212._dac_buffer_config.triggerMode (*C var*), 190
drivers_Kinetis_K_MK22F51212._dac_buffer_config.upperLimit (*C var*), 190
drivers_Kinetis_K_MK22F51212._dac_buffer_config.watermark (*C var*), 190
drivers_Kinetis_K_MK22F51212._dac_buffer_config.workMode (*C var*), 190
drivers_Kinetis_K_MK22F51212._dac_buffer_interrupt_enable (*C enum*), 188
drivers_Kinetis_K_MK22F51212._dac_buffer_interrupt_enable.kDAC_BufferReadPointerBottomInterruptEnable (*C enumerator*), 188
drivers_Kinetis_K_MK22F51212._dac_buffer_interrupt_enable.kDAC_BufferReadPointerTopInterruptEnable (*C enumerator*), 188
drivers_Kinetis_K_MK22F51212._dac_buffer_interrupt_enable.kDAC_BufferWatermarkInterruptEnable (*C enumerator*), 188
drivers_Kinetis_K_MK22F51212._dac_buffer_status_flags (*C enum*), 188
drivers_Kinetis_K_MK22F51212._dac_buffer_status_flags.kDAC_BufferReadPointerBottomPositionFlag (*C enumerator*), 188
drivers_Kinetis_K_MK22F51212._dac_buffer_status_flags.kDAC_BufferReadPointerTopPositionFlag (*C enumerator*), 188
drivers_Kinetis_K_MK22F51212._dac_buffer_status_flags.kDAC_BufferWatermarkFlag (*C enumerator*), 188
drivers_Kinetis_K_MK22F51212._dac_buffer_trigger_mode (*C enum*), 188
drivers_Kinetis_K_MK22F51212._dac_buffer_trigger_mode.kDAC_BufferTriggerByHardwareMode (*C enumerator*), 188
drivers_Kinetis_K_MK22F51212._dac_buffer_trigger_mode.kDAC_BufferTriggerBySoftwareMode (*C enumerator*), 188
drivers_Kinetis_K_MK22F51212._dac_buffer_watermark (*C enum*), 189
drivers_Kinetis_K_MK22F51212._dac_buffer_watermark.kDAC_BufferWatermark1Word (*C enumerator*), 189
drivers_Kinetis_K_MK22F51212._dac_buffer_watermark.kDAC_BufferWatermark2Word (*C enumerator*), 189

drivers_Kinetis_K_MK22F51212._dac_buffer_watermark.kDAC_BufferWatermark3Word (*C enumerator*), 189
drivers_Kinetis_K_MK22F51212._dac_buffer_watermark.kDAC_BufferWatermark4Word (*C enumerator*), 189
drivers_Kinetis_K_MK22F51212._dac_buffer_work_mode (*C enum*), 189
drivers_Kinetis_K_MK22F51212._dac_buffer_work_mode.kDAC_BufferWorkAsFIFOMode (*C enumerator*), 189
drivers_Kinetis_K_MK22F51212._dac_buffer_work_mode.kDAC_BufferWorkAsNormalMode (*C enumerator*), 189
drivers_Kinetis_K_MK22F51212._dac_buffer_work_mode.kDAC_BufferWorkAsOneTimeScanMode (*C enumerator*), 189
drivers_Kinetis_K_MK22F51212._dac_buffer_work_mode.kDAC_BufferWorkAsSwingMode (*C enumerator*), 189
drivers_Kinetis_K_MK22F51212._dac_config (*C struct*), 190
drivers_Kinetis_K_MK22F51212._dac_config.enableLowPowerMode (*C var*), 190
drivers_Kinetis_K_MK22F51212._dac_config.referenceVoltageSource (*C var*), 190
drivers_Kinetis_K_MK22F51212._dac_reference_voltage_source (*C enum*), 188
drivers_Kinetis_K_MK22F51212._dac_reference_voltage_source.kDAC_ReferenceVoltageSourceVref1 (*C enumerator*), 188
drivers_Kinetis_K_MK22F51212._dac_reference_voltage_source.kDAC_ReferenceVoltageSourceVref2 (*C enumerator*), 188
drivers_Kinetis_K_MK22F51212._dspi_clock_phase (*C enum*), 208
drivers_Kinetis_K_MK22F51212._dspi_clock_phase.kDSPI_ClockPhaseFirstEdge (*C enumerator*), 208
drivers_Kinetis_K_MK22F51212._dspi_clock_phase.kDSPI_ClockPhaseSecondEdge (*C enumerator*), 208
drivers_Kinetis_K_MK22F51212._dspi_clock_polarity (*C enum*), 208
drivers_Kinetis_K_MK22F51212._dspi_clock_polarity.kDSPI_ClockPolarityActiveHigh (*C enumerator*), 208
drivers_Kinetis_K_MK22F51212._dspi_clock_polarity.kDSPI_ClockPolarityActiveLow (*C enumerator*), 208
drivers_Kinetis_K_MK22F51212._dspi_command_data_config (*C struct*), 213
drivers_Kinetis_K_MK22F51212._dspi_command_data_config.clearTransferCount (*C var*), 213
drivers_Kinetis_K_MK22F51212._dspi_command_data_config.isEndOfQueue (*C var*), 213
drivers_Kinetis_K_MK22F51212._dspi_command_data_config.isPcsContinuous (*C var*), 213
drivers_Kinetis_K_MK22F51212._dspi_command_data_config.whichCtar (*C var*), 213
drivers_Kinetis_K_MK22F51212._dspi_command_data_config.whichPcs (*C var*), 213
drivers_Kinetis_K_MK22F51212._dspi_ctar_selection (*C enum*), 209
drivers_Kinetis_K_MK22F51212._dspi_ctar_selection.kDSPI_Ctar0 (*C enumerator*), 209
drivers_Kinetis_K_MK22F51212._dspi_ctar_selection.kDSPI_Ctar1 (*C enumerator*), 209
drivers_Kinetis_K_MK22F51212._dspi_ctar_selection.kDSPI_Ctar2 (*C enumerator*), 209
drivers_Kinetis_K_MK22F51212._dspi_ctar_selection.kDSPI_Ctar3 (*C enumerator*), 209
drivers_Kinetis_K_MK22F51212._dspi_ctar_selection.kDSPI_Ctar4 (*C enumerator*), 209
drivers_Kinetis_K_MK22F51212._dspi_ctar_selection.kDSPI_Ctar5 (*C enumerator*), 209
drivers_Kinetis_K_MK22F51212._dspi_ctar_selection.kDSPI_Ctar6 (*C enumerator*), 209
drivers_Kinetis_K_MK22F51212._dspi_ctar_selection.kDSPI_Ctar7 (*C enumerator*), 209
drivers_Kinetis_K_MK22F51212._dspi_delay_type (*C enum*), 208
drivers_Kinetis_K_MK22F51212._dspi_delay_type.kDSPI_BetweenTransfer (*C enumerator*), 209
drivers_Kinetis_K_MK22F51212._dspi_delay_type.kDSPI_LastSckToPcs (*C enumerator*), 209
drivers_Kinetis_K_MK22F51212._dspi_delay_type.kDSPI_PcsToSck (*C enumerator*), 208
drivers_Kinetis_K_MK22F51212._dspi_dma_enable (*C enum*), 206
drivers_Kinetis_K_MK22F51212._dspi_dma_enable.kDSPI_RxDmaEnable (*C enumerator*), 206
drivers_Kinetis_K_MK22F51212._dspi_dma_enable.kDSPI_TxDmaEnable (*C enumerator*), 206
drivers_Kinetis_K_MK22F51212._dspi_flags (*C enum*), 205
drivers_Kinetis_K_MK22F51212._dspi_flags.kDSPI_AllStatusFlag (*C enumerator*), 205
drivers_Kinetis_K_MK22F51212._dspi_flags.kDSPI_EndOfQueueFlag (*C enumerator*), 205
drivers_Kinetis_K_MK22F51212._dspi_flags.kDSPI_RxFifoDrainRequestFlag (*C enumerator*), 205
drivers_Kinetis_K_MK22F51212._dspi_flags.kDSPI_RxFifoOverflowFlag (*C enumerator*), 205

drivers_Kinetis_K_MK22F51212._dspi_flags.kDSPI_TxAndRxStatusFlag (*C enumerator*), 205
drivers_Kinetis_K_MK22F51212._dspi_flags.kDSPI_TxCompleteFlag (*C enumerator*), 205
drivers_Kinetis_K_MK22F51212._dspi_flags.kDSPI_TxFifoFillRequestFlag (*C enumerator*), 205
drivers_Kinetis_K_MK22F51212._dspi_flags.kDSPI_TxFifoUnderflowFlag (*C enumerator*), 205
drivers_Kinetis_K_MK22F51212._dspi_half_duplex_transfer (*C struct*), 216
drivers_Kinetis_K_MK22F51212._dspi_half_duplex_transfer.configFlags (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_half_duplex_transfer.isPcsAssertInTransfer (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_half_duplex_transfer.isTransmitFirst (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_half_duplex_transfer.rxData (*C var*), 216
drivers_Kinetis_K_MK22F51212._dspi_half_duplex_transfer.rxDataSize (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_half_duplex_transfer.txData (*C var*), 216
drivers_Kinetis_K_MK22F51212._dspi_half_duplex_transfer.txDataSize (*C var*), 216
drivers_Kinetis_K_MK22F51212._dspi_interrupt_enable (*C enum*), 205
drivers_Kinetis_K_MK22F51212._dspi_interrupt_enable.kDSPI_AllInterruptEnable (*C enumerator*), 206
drivers_Kinetis_K_MK22F51212._dspi_interrupt_enable.kDSPI_EndOfQueueInterruptEnable (*C enumerator*), 205
drivers_Kinetis_K_MK22F51212._dspi_interrupt_enable.kDSPI_RxFifoDrainRequestInterruptEnable (*C enumerator*), 206
drivers_Kinetis_K_MK22F51212._dspi_interrupt_enable.kDSPI_RxFifoOverflowInterruptEnable (*C enumerator*), 206
drivers_Kinetis_K_MK22F51212._dspi_interrupt_enable.kDSPI_TxCompleteInterruptEnable (*C enumerator*), 205
drivers_Kinetis_K_MK22F51212._dspi_interrupt_enable.kDSPI_TxFifoFillRequestInterruptEnable (*C enumerator*), 206
drivers_Kinetis_K_MK22F51212._dspi_interrupt_enable.kDSPI_TxFifoUnderflowInterruptEnable (*C enumerator*), 206
drivers_Kinetis_K_MK22F51212._dspi_master_config (*C struct*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_config.ctarConfig (*C var*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_config.enableContinuousSCK (*C var*), 215
drivers_Kinetis_K_MK22F51212._dspi_master_config.enableModifiedTimingFormat (*C var*), 215
drivers_Kinetis_K_MK22F51212._dspi_master_config.enableRxFifoOverWrite (*C var*), 215
drivers_Kinetis_K_MK22F51212._dspi_master_config.pcsActiveHighOrLow (*C var*), 215
drivers_Kinetis_K_MK22F51212._dspi_master_config.samplePoint (*C var*), 215
drivers_Kinetis_K_MK22F51212._dspi_master_config.whichCtar (*C var*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_config.whichPcs (*C var*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_ctar_config (*C struct*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_ctar_config.baudRate (*C var*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_ctar_config.betweenTransferDelayInNanoSec (*C var*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_ctar_config.bitsPerFrame (*C var*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_ctar_config.cpha (*C var*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_ctar_config.cpol (*C var*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_ctar_config.direction (*C var*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_ctar_config.lastSckToPcsDelayInNanoSec (*C var*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_ctar_config.pcsToSckDelayInNanoSec (*C var*), 214
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle (*C struct*), 223
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.bitsPerFrame (*C var*), 223
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.callback (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.command (*C var*), 223
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.dspiSoftwareTCD (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.edmaIntermediaryToTxRegHandle (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.edmaRxRegToRxDataHandle (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.edmaTxDataToIntermediaryHandle (*C var*), 224

drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.fifoSize (*C var*), 223
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.isPcsActiveAfterTransfer (*C var*), 223
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.lastCommand (*C var*), 223
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.nbytes (*C var*), 223
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.remainingReceiveByteCount (*C var*), 223
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.remainingSendByteCount (*C var*), 223
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.rxBuffIfNull (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.rxData (*C var*), 223
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.state (*C var*), 223
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.totalByteCount (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.txBuffIfNull (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.txData (*C var*), 223
drivers_Kinetis_K_MK22F51212._dspi_master_edma_handle.userData (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_master_handle (*C struct*), 217
drivers_Kinetis_K_MK22F51212._dspi_master_handle.bitsPerFrame (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_master_handle.callback (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_master_handle.command (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_master_handle.fifoSize (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_master_handle.isPcsActiveAfterTransfer (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_master_handle.isThereExtraByte (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_master_handle.lastCommand (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_master_handle.remainingReceiveByteCount (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_master_handle.remainingSendByteCount (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_master_handle.rxData (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_master_handle.state (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_master_handle.totalByteCount (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_master_handle.txData (*C var*), 217
drivers_Kinetis_K_MK22F51212._dspi_master_handle.userData (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_master_sample_point (*C enum*), 206
drivers_Kinetis_K_MK22F51212._dspi_master_sample_point.kDSPI_SckToSin0Clock (*C enumerator*), 206
drivers_Kinetis_K_MK22F51212._dspi_master_sample_point.kDSPI_SckToSin1Clock (*C enumerator*), 206
drivers_Kinetis_K_MK22F51212._dspi_master_sample_point.kDSPI_SckToSin2Clock (*C enumerator*), 206
drivers_Kinetis_K_MK22F51212._dspi_master_slave_mode (*C enum*), 206
drivers_Kinetis_K_MK22F51212._dspi_master_slave_mode.kDSPI_Master (*C enumerator*), 206
drivers_Kinetis_K_MK22F51212._dspi_master_slave_mode.kDSPI_Slave (*C enumerator*), 206
drivers_Kinetis_K_MK22F51212._dspi_pcs_polarity (*C enum*), 207
drivers_Kinetis_K_MK22F51212._dspi_pcs_polarity_config (*C enum*), 207
drivers_Kinetis_K_MK22F51212._dspi_pcs_polarity_config.kDSPI_PcsActiveHigh (*C enumerator*), 207
drivers_Kinetis_K_MK22F51212._dspi_pcs_polarity_config.kDSPI_PcsActiveLow (*C enumerator*), 207
drivers_Kinetis_K_MK22F51212._dspi_pcs_polarity.kDSPI_Pcs0ActiveLow (*C enumerator*), 207
drivers_Kinetis_K_MK22F51212._dspi_pcs_polarity.kDSPI_Pcs1ActiveLow (*C enumerator*), 207
drivers_Kinetis_K_MK22F51212._dspi_pcs_polarity.kDSPI_Pcs2ActiveLow (*C enumerator*), 207
drivers_Kinetis_K_MK22F51212._dspi_pcs_polarity.kDSPI_Pcs3ActiveLow (*C enumerator*), 207
drivers_Kinetis_K_MK22F51212._dspi_pcs_polarity.kDSPI_Pcs4ActiveLow (*C enumerator*), 208
drivers_Kinetis_K_MK22F51212._dspi_pcs_polarity.kDSPI_Pcs5ActiveLow (*C enumerator*), 208
drivers_Kinetis_K_MK22F51212._dspi_pcs_polarity.kDSPI_PcsAllActiveLow (*C enumerator*), 208
drivers_Kinetis_K_MK22F51212._dspi_shift_direction (*C enum*), 208
drivers_Kinetis_K_MK22F51212._dspi_shift_direction.kDSPI_LsbFirst (*C enumerator*), 208
drivers_Kinetis_K_MK22F51212._dspi_shift_direction.kDSPI_MsbFirst (*C enumerator*), 208
drivers_Kinetis_K_MK22F51212._dspi_slave_config (*C struct*), 215
drivers_Kinetis_K_MK22F51212._dspi_slave_config.ctarConfig (*C var*), 215
drivers_Kinetis_K_MK22F51212._dspi_slave_config.enableContinuousSCK (*C var*), 215

drivers_Kinetis_K_MK22F51212._dspi_slave_config.enableModifiedTimingFormat (*C var*), 216
drivers_Kinetis_K_MK22F51212._dspi_slave_config.enableRxFifoOverWrite (*C var*), 216
drivers_Kinetis_K_MK22F51212._dspi_slave_config.samplePoint (*C var*), 216
drivers_Kinetis_K_MK22F51212._dspi_slave_config.whichCtar (*C var*), 215
drivers_Kinetis_K_MK22F51212._dspi_slave_ctar_config (*C struct*), 215
drivers_Kinetis_K_MK22F51212._dspi_slave_ctar_config.bitsPerFrame (*C var*), 215
drivers_Kinetis_K_MK22F51212._dspi_slave_ctar_config.cpha (*C var*), 215
drivers_Kinetis_K_MK22F51212._dspi_slave_ctar_config.cpol (*C var*), 215
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle (*C struct*), 224
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.bitsPerFrame (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.callback (*C var*), 225
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.edmaRxRegToRxDataHandle (*C var*), 225
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.edmaTxDataToTxRegHandle (*C var*), 225
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.nbytes (*C var*), 225
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.remainingReceiveByteCount (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.remainingSendByteCount (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.rxBuffIfNull (*C var*), 225
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.rxData (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.state (*C var*), 225
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.totalByteCount (*C var*), 225
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.txBuffIfNull (*C var*), 225
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.txData (*C var*), 224
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.txLastData (*C var*), 225
drivers_Kinetis_K_MK22F51212._dspi_slave_edma_handle.userData (*C var*), 225
drivers_Kinetis_K_MK22F51212._dspi_slave_handle (*C struct*), 218
drivers_Kinetis_K_MK22F51212._dspi_slave_handle.bitsPerFrame (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_slave_handle.callback (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_slave_handle.errorCount (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_slave_handle.isThereExtraByte (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_slave_handle.remainingReceiveByteCount (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_slave_handle.remainingSendByteCount (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_slave_handle.rxData (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_slave_handle.state (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_slave_handle.totalByteCount (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_slave_handle.txData (*C var*), 218
drivers_Kinetis_K_MK22F51212._dspi_slave_handle.userData (*C var*), 219
drivers_Kinetis_K_MK22F51212._dspi_transfer (*C struct*), 216
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master (*C enum*), 209
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterActiveAfterTransfer (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterCtar0 (*C enumerator*), 209
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterCtar1 (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterCtar2 (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterCtar3 (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterCtar4 (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterCtar5 (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterCtar6 (*C enumerator*), 210

drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterCtar7 (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterPcs0 (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterPcs1 (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterPcs2 (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterPcs3 (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterPcs4 (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterPcs5 (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_master.kDSPI_MasterPcsContinuous (*C enumerator*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_slave (*C enum*), 210
drivers_Kinetis_K_MK22F51212._dspi_transfer_config_flag_for_slave.kDSPI_SlaveCtar0 (*C enumerator*), 211
drivers_Kinetis_K_MK22F51212._dspi_transfer_state (*C enum*), 211
drivers_Kinetis_K_MK22F51212._dspi_transfer_state.kDSPI_Busy (*C enumerator*), 211
drivers_Kinetis_K_MK22F51212._dspi_transfer_state.kDSPI_Error (*C enumerator*), 211
drivers_Kinetis_K_MK22F51212._dspi_transfer_state.kDSPI_Idle (*C enumerator*), 211
drivers_Kinetis_K_MK22F51212._dspi_transfer.configFlags (*C var*), 216
drivers_Kinetis_K_MK22F51212._dspi_transfer.dataSize (*C var*), 216
drivers_Kinetis_K_MK22F51212._dspi_transfer.rxData (*C var*), 216
drivers_Kinetis_K_MK22F51212._dspi_transfer.txData (*C var*), 216
drivers_Kinetis_K_MK22F51212._dspi_which_pcs_config (*C enum*), 207
drivers_Kinetis_K_MK22F51212._dspi_which_pcs_config.kDSPI_Pcs0 (*C enumerator*), 207
drivers_Kinetis_K_MK22F51212._dspi_which_pcs_config.kDSPI_Pcs1 (*C enumerator*), 207
drivers_Kinetis_K_MK22F51212._dspi_which_pcs_config.kDSPI_Pcs2 (*C enumerator*), 207
drivers_Kinetis_K_MK22F51212._dspi_which_pcs_config.kDSPI_Pcs3 (*C enumerator*), 207
drivers_Kinetis_K_MK22F51212._dspi_which_pcs_config.kDSPI_Pcs4 (*C enumerator*), 207
drivers_Kinetis_K_MK22F51212._dspi_which_pcs_config.kDSPI_Pcs5 (*C enumerator*), 207
drivers_Kinetis_K_MK22F51212._edma_bandwidth (*C enum*), 240
drivers_Kinetis_K_MK22F51212._edma_bandwidth.kEDMA_BandwidthStall4Cycle (*C enumerator*), 240
drivers_Kinetis_K_MK22F51212._edma_bandwidth.kEDMA_BandwidthStall8Cycle (*C enumerator*), 240
drivers_Kinetis_K_MK22F51212._edma_bandwidth.kEDMA_BandwidthStallNone (*C enumerator*), 240
drivers_Kinetis_K_MK22F51212._edma_channel_link_type (*C enum*), 240
drivers_Kinetis_K_MK22F51212._edma_channel_link_type.kEDMA_LinkNone (*C enumerator*), 240
drivers_Kinetis_K_MK22F51212._edma_channel_link_type.kEDMA_MajorLink (*C enumerator*), 240
drivers_Kinetis_K_MK22F51212._edma_channel_link_type.kEDMA_MinorLink (*C enumerator*), 240
drivers_Kinetis_K_MK22F51212._edma_channel_Preemption_config (*C struct*), 245
drivers_Kinetis_K_MK22F51212._edma_channel_Preemption_config.channelPriority (*C var*), 245
drivers_Kinetis_K_MK22F51212._edma_channel_Preemption_config.enableChannelPreemption (*C var*), 245
drivers_Kinetis_K_MK22F51212._edma_channel_Preemption_config.enablePreemptAbility (*C var*), 245
drivers_Kinetis_K_MK22F51212._edma_config (*C struct*), 243
drivers_Kinetis_K_MK22F51212._edma_config.enableContinuousLinkMode (*C var*), 244
drivers_Kinetis_K_MK22F51212._edma_config.enableDebugMode (*C var*), 244
drivers_Kinetis_K_MK22F51212._edma_config.enableHaltOnError (*C var*), 244

drivers_Kinetis_K_MK22F51212._edma_config.enableRoundRobinArbitration (*C var*), 244
drivers_Kinetis_K_MK22F51212._edma_handle (*C struct*), 246
drivers_Kinetis_K_MK22F51212._edma_handle.base (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_handle.callback (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_handle.channel (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_handle.flags (*C var*), 247
drivers_Kinetis_K_MK22F51212._edma_handle.header (*C var*), 247
drivers_Kinetis_K_MK22F51212._edma_handle.tail (*C var*), 247
drivers_Kinetis_K_MK22F51212._edma_handle.tcdPool (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_handle.tcdSize (*C var*), 247
drivers_Kinetis_K_MK22F51212._edma_handle.tcdUsed (*C var*), 247
drivers_Kinetis_K_MK22F51212._edma_handle.userData (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_interrupt_enable (*C enum*), 241
drivers_Kinetis_K_MK22F51212._edma_interrupt_enable.kEDMA_ErrorInterruptEnable (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212._edma_interrupt_enable.kEDMA_HalfInterruptEnable (*C enumerator*), 242
drivers_Kinetis_K_MK22F51212._edma_interrupt_enable.kEDMA_MajorInterruptEnable (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212._edma_minor_offset_config (*C struct*), 245
drivers_Kinetis_K_MK22F51212._edma_minor_offset_config.enableDestMinorOffset (*C var*), 245
drivers_Kinetis_K_MK22F51212._edma_minor_offset_config.enableSrcMinorOffset (*C var*), 245
drivers_Kinetis_K_MK22F51212._edma_minor_offset_config.minorOffset (*C var*), 245
drivers_Kinetis_K_MK22F51212._edma_modulo (*C enum*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo1Gbytes (*C enumerator*), 240
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo1Kbytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo1Mbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo2bytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo2Gbytes (*C enumerator*), 240
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo2Kbytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo2Mbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo4bytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo4Kbytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo4Mbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo8bytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo8Kbytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo8Mbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo16bytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo16Kbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo16Mbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo32bytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo32Kbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo32Mbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo64bytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo64Kbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo64Mbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo128bytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo128Kbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo128Mbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo256bytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo256Kbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo256Mbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo512bytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo512Kbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_Modulo512Mbytes (*C enumerator*), 239
drivers_Kinetis_K_MK22F51212._edma_modulo.kEDMA_ModuloDisable (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_tcd (*C struct*), 245
drivers_Kinetis_K_MK22F51212._edma_tcd.ATTR (*C var*), 246

drivers_Kinetis_K_MK22F51212._edma_tcd.BITER (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_tcd.CITER (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_tcd.CSR (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_tcd.DADDR (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_tcd.DLAST_SGA (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_tcd.DOFF (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_tcd.NBYTES (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_tcd.SADDR (*C var*), 245
drivers_Kinetis_K_MK22F51212._edma_tcd.SLAST (*C var*), 246
drivers_Kinetis_K_MK22F51212._edma_tcd.SOFF (*C var*), 245
drivers_Kinetis_K_MK22F51212._edma_transfer_config (*C struct*), 244
drivers_Kinetis_K_MK22F51212._edma_transfer_config.destAddr (*C var*), 244
drivers_Kinetis_K_MK22F51212._edma_transfer_config.destOffset (*C var*), 244
drivers_Kinetis_K_MK22F51212._edma_transfer_config.destTransferSize (*C var*), 244
drivers_Kinetis_K_MK22F51212._edma_transfer_config.majorLoopCounts (*C var*), 245
drivers_Kinetis_K_MK22F51212._edma_transfer_config.minorLoopBytes (*C var*), 244
drivers_Kinetis_K_MK22F51212._edma_transfer_config.srcAddr (*C var*), 244
drivers_Kinetis_K_MK22F51212._edma_transfer_config.srcOffset (*C var*), 244
drivers_Kinetis_K_MK22F51212._edma_transfer_config.srcTransferSize (*C var*), 244
drivers_Kinetis_K_MK22F51212._edma_transfer_size (*C enum*), 237
drivers_Kinetis_K_MK22F51212._edma_transfer_size.kEDMA_TransferSize1Bytes (*C enumerator*), 237
drivers_Kinetis_K_MK22F51212._edma_transfer_size.kEDMA_TransferSize2Bytes (*C enumerator*), 237
drivers_Kinetis_K_MK22F51212._edma_transfer_size.kEDMA_TransferSize4Bytes (*C enumerator*), 237
drivers_Kinetis_K_MK22F51212._edma_transfer_size.kEDMA_TransferSize8Bytes (*C enumerator*), 237
drivers_Kinetis_K_MK22F51212._edma_transfer_size.kEDMA_TransferSize16Bytes (*C enumerator*), 237
drivers_Kinetis_K_MK22F51212._edma_transfer_size.kEDMA_TransferSize32Bytes (*C enumerator*), 238
drivers_Kinetis_K_MK22F51212._edma_transfer_type (*C enum*), 242
drivers_Kinetis_K_MK22F51212._edma_transfer_type.kEDMA_MemoryToMemory (*C enumerator*), 242
drivers_Kinetis_K_MK22F51212._edma_transfer_type.kEDMA_MemoryToPeripheral (*C enumerator*), 242
drivers_Kinetis_K_MK22F51212._edma_transfer_type.kEDMA_PeripheralToMemory (*C enumerator*), 242
drivers_Kinetis_K_MK22F51212._edma_transfer_type.kEDMA_PeripheralToPeripheral (*C enumerator*), 242
drivers_Kinetis_K_MK22F51212._ewm_config (*C struct*), 250
drivers_Kinetis_K_MK22F51212._ewm_config.clockSource (*C var*), 250
drivers_Kinetis_K_MK22F51212._ewm_config.compareHighValue (*C var*), 250
drivers_Kinetis_K_MK22F51212._ewm_config.compareLowValue (*C var*), 250
drivers_Kinetis_K_MK22F51212._ewm_config.enableEwm (*C var*), 250
drivers_Kinetis_K_MK22F51212._ewm_config.enableEwmInput (*C var*), 250
drivers_Kinetis_K_MK22F51212._ewm_config.enableInterrupt (*C var*), 250
drivers_Kinetis_K_MK22F51212._ewm_config.prescaler (*C var*), 250
drivers_Kinetis_K_MK22F51212._ewm_config.setInputAssertLogic (*C var*), 250
drivers_Kinetis_K_MK22F51212._ewm_interrupt_enable_t (*C enum*), 249
drivers_Kinetis_K_MK22F51212._ewm_interrupt_enable_t.kEWM_InterruptEnable (*C enumerator*), 249
drivers_Kinetis_K_MK22F51212._ewm_lpo_clock_source (*C enum*), 249
drivers_Kinetis_K_MK22F51212._ewm_lpo_clock_source.kEWM_LpoClockSource0 (*C enumerator*), 249
drivers_Kinetis_K_MK22F51212._ewm_lpo_clock_source.kEWM_LpoClockSource1 (*C enumerator*), 249

drivers_Kinetis_K_MK22F51212._ewm_lpo_clock_source.kEWM_LpoClockSource2 (*C enumerator*), 249
drivers_Kinetis_K_MK22F51212._ewm_lpo_clock_source.kEWM_LpoClockSource3 (*C enumerator*), 249
drivers_Kinetis_K_MK22F51212._ewm_status_flags_t (*C enum*), 249
drivers_Kinetis_K_MK22F51212._ewm_status_flags_t.kEWM_RunningFlag (*C enumerator*), 249
drivers_Kinetis_K_MK22F51212._flash_acceleration_ram_property (*C enum*), 272
drivers_Kinetis_K_MK22F51212._flash_acceleration_ram_property.kFLASH_AccelerationRamSize (*C enumerator*), 272
drivers_Kinetis_K_MK22F51212._flash_config (*C struct*), 291
drivers_Kinetis_K_MK22F51212._flash_execute_only_access_state (*C enum*), 289
drivers_Kinetis_K_MK22F51212._flash_execute_only_access_state.kFLASH_AccessStateExecuteOnly (*C enumerator*), 289
drivers_Kinetis_K_MK22F51212._flash_execute_only_access_state.kFLASH_AccessStateMixed (*C enumerator*), 289
drivers_Kinetis_K_MK22F51212._flash_execute_only_access_state.kFLASH_AccessStateUnLimited (*C enumerator*), 289
drivers_Kinetis_K_MK22F51212._flash_prefetch_speculation_status (*C struct*), 258
drivers_Kinetis_K_MK22F51212._flash_prefetch_speculation_status.dataOff (*C var*), 258
drivers_Kinetis_K_MK22F51212._flash_prefetch_speculation_status.instructionOff (*C var*), 258
drivers_Kinetis_K_MK22F51212._flash_property_tag (*C enum*), 289
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyFlexRamBlockBaseAddr (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyFlexRamTotalSize (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash0AccessSegmentCount (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash0AccessSegmentSize (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash0BlockBaseAddr (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash0BlockCount (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash0BlockSize (*C enumerator*), 289
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash0FacSupport (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash0SectorSize (*C enumerator*), 289
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash0TotalSize (*C enumerator*), 289
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash1AccessSegmentCount (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash1AccessSegmentSize (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash1BlockBaseAddr (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash1BlockCount (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash1BlockSize (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash1FacSupport (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash1SectorSize (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_property_tag.kFLASH_PropertyPflash1TotalSize (*C enumerator*), 290
drivers_Kinetis_K_MK22F51212._flash_protection_state (*C enum*), 289

drivers_Kinetis_K_MK22F51212._flash_protection_state.kFLASH_ProtectionStateMixed (*C enumerator*), 289
drivers_Kinetis_K_MK22F51212._flash_protection_state.kFLASH_ProtectionStateProtected (*C enumerator*), 289
drivers_Kinetis_K_MK22F51212._flash_protection_state.kFLASH_ProtectionStateUnprotected (*C enumerator*), 289
drivers_Kinetis_K_MK22F51212._flexbus_address_setup (*C enum*), 252
drivers_Kinetis_K_MK22F51212._flexbus_address_setup.kFLEXBUS_FirstRisingEdge (*C enumerator*), 252
drivers_Kinetis_K_MK22F51212._flexbus_address_setup.kFLEXBUS_FourthRisingEdge (*C enumerator*), 253
drivers_Kinetis_K_MK22F51212._flexbus_address_setup.kFLEXBUS_SecondRisingEdge (*C enumerator*), 253
drivers_Kinetis_K_MK22F51212._flexbus_address_setup.kFLEXBUS_ThirdRisingEdge (*C enumerator*), 253
drivers_Kinetis_K_MK22F51212._flexbus_bytelane_shift (*C enum*), 253
drivers_Kinetis_K_MK22F51212._flexbus_bytelane_shift.kFLEXBUS_NotShifted (*C enumerator*), 253
drivers_Kinetis_K_MK22F51212._flexbus_bytelane_shift.kFLEXBUS_Shifted (*C enumerator*), 253
drivers_Kinetis_K_MK22F51212._flexbus_config (*C struct*), 255
drivers_Kinetis_K_MK22F51212._flexbus_config.addressSetup (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.autoAcknowledge (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.burstRead (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.burstWrite (*C var*), 255
drivers_Kinetis_K_MK22F51212._flexbus_config.byteEnableMode (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.byteLaneShift (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.chip (*C var*), 255
drivers_Kinetis_K_MK22F51212._flexbus_config.chipBaseAddress (*C var*), 255
drivers_Kinetis_K_MK22F51212._flexbus_config.chipBaseAddressMask (*C var*), 255
drivers_Kinetis_K_MK22F51212._flexbus_config.extendTransferAddress (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.group1MultiplexControl (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.group2MultiplexControl (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.group3MultiplexControl (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.group4MultiplexControl (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.group5MultiplexControl (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.portSize (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.readAddressHold (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.secondaryWaitStates (*C var*), 255
drivers_Kinetis_K_MK22F51212._flexbus_config.secondaryWaitStatesEnable (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.waitStates (*C var*), 255
drivers_Kinetis_K_MK22F51212._flexbus_config.writeAddressHold (*C var*), 256
drivers_Kinetis_K_MK22F51212._flexbus_config.writeProtect (*C var*), 255
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group1_signal (*C enum*), 253
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group1_signal.kFLEXBUS_MultiplexGroup1_FB_ALE (*C enumerator*), 253
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group1_signal.kFLEXBUS_MultiplexGroup1_FB_CS1 (*C enumerator*), 253
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group1_signal.kFLEXBUS_MultiplexGroup1_FB_TS (*C enumerator*), 253
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group2_signal (*C enum*), 253
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group2_signal.kFLEXBUS_MultiplexGroup2_FB_BE_31_24 (*C enumerator*), 253
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group2_signal.kFLEXBUS_MultiplexGroup2_FB_CS4 (*C enumerator*), 253
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group2_signal.kFLEXBUS_MultiplexGroup2_FB_TSIZ0 (*C enumerator*), 253
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group3_signal (*C enum*), 253
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group3_signal.kFLEXBUS_MultiplexGroup3_FB_BE_23_16

(*C enumerator*), 254
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group3_signal.kFLEXBUS_MultiplexGroup3_FB_CS5 (*C enumerator*), 254
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group3_signal.kFLEXBUS_MultiplexGroup3_FB_TSIZ1 (*C enumerator*), 254
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group4_signal (*C enum*), 254
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group4_signal.kFLEXBUS_MultiplexGroup4_FB_BE_15_8 (*C enumerator*), 254
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group4_signal.kFLEXBUS_MultiplexGroup4_FB_CS2 (*C enumerator*), 254
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group4_signal.kFLEXBUS_MultiplexGroup4_FB_TBST (*C enumerator*), 254
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group5_signal (*C enum*), 254
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group5_signal.kFLEXBUS_MultiplexGroup5_FB_BE_7_0 (*C enumerator*), 254
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group5_signal.kFLEXBUS_MultiplexGroup5_FB_CS3 (*C enumerator*), 254
drivers_Kinetis_K_MK22F51212._flexbus_multiplex_group5_signal.kFLEXBUS_MultiplexGroup5_FB_TA (*C enumerator*), 254
drivers_Kinetis_K_MK22F51212._flexbus_port_size (*C enum*), 251
drivers_Kinetis_K_MK22F51212._flexbus_port_size.kFLEXBUS_1Byte (*C enumerator*), 252
drivers_Kinetis_K_MK22F51212._flexbus_port_size.kFLEXBUS_2Bytes (*C enumerator*), 252
drivers_Kinetis_K_MK22F51212._flexbus_port_size.kFLEXBUS_4Bytes (*C enumerator*), 252
drivers_Kinetis_K_MK22F51212._flexbus_read_address_hold (*C enum*), 252
drivers_Kinetis_K_MK22F51212._flexbus_read_address_hold.kFLEXBUS_Hold1Or0Cycles (*C enumerator*), 252
drivers_Kinetis_K_MK22F51212._flexbus_read_address_hold.kFLEXBUS_Hold2Or1Cycles (*C enumerator*), 252
drivers_Kinetis_K_MK22F51212._flexbus_read_address_hold.kFLEXBUS_Hold3Or2Cycle (*C enumerator*), 252
drivers_Kinetis_K_MK22F51212._flexbus_read_address_hold.kFLEXBUS_Hold4Or3Cycle (*C enumerator*), 252
drivers_Kinetis_K_MK22F51212._flexbus_write_address_hold (*C enum*), 252
drivers_Kinetis_K_MK22F51212._flexbus_write_address_hold.kFLEXBUS_Hold1Cycle (*C enumerator*), 252
drivers_Kinetis_K_MK22F51212._flexbus_write_address_hold.kFLEXBUS_Hold2Cycles (*C enumerator*), 252
drivers_Kinetis_K_MK22F51212._flexbus_write_address_hold.kFLEXBUS_Hold3Cycles (*C enumerator*), 252
drivers_Kinetis_K_MK22F51212._flexbus_write_address_hold.kFLEXBUS_Hold4Cycles (*C enumerator*), 252
drivers_Kinetis_K_MK22F51212._flexnvm_config (*C struct*), 303
drivers_Kinetis_K_MK22F51212._flexnvm_property_tag (*C enum*), 301
drivers_Kinetis_K_MK22F51212._flexnvm_property_tag.kFLEXNVM_PropertyAliasDflashBlockBaseAddr (*C enumerator*), 302
drivers_Kinetis_K_MK22F51212._flexnvm_property_tag.kFLEXNVM_PropertyDflashBlockBaseAddr (*C enumerator*), 302
drivers_Kinetis_K_MK22F51212._flexnvm_property_tag.kFLEXNVM_PropertyDflashBlockCount (*C enumerator*), 301
drivers_Kinetis_K_MK22F51212._flexnvm_property_tag.kFLEXNVM_PropertyDflashBlockSize (*C enumerator*), 301
drivers_Kinetis_K_MK22F51212._flexnvm_property_tag.kFLEXNVM_PropertyDflashSectorSize (*C enumerator*), 301
drivers_Kinetis_K_MK22F51212._flexnvm_property_tag.kFLEXNVM_PropertyDflashTotalSize (*C enumerator*), 301
drivers_Kinetis_K_MK22F51212._flexnvm_property_tag.kFLEXNVM_PropertyEepromTotalSize (*C enumerator*), 302
drivers_Kinetis_K_MK22F51212._flexnvm_property_tag.kFLEXNVM_PropertyFlexRamBlockBaseAddr

(*C enumerator*), 302
drivers_Kinetis_K_MK22F51212._flexnvm_property_tag.kFLEXNVM_PropertyFlexRamTotalSize (*C enumerator*), 302
drivers_Kinetis_K_MK22F51212._ftfx_cache_config (*C struct*), 258
drivers_Kinetis_K_MK22F51212._ftfx_cache_config.bitOperFuncAddr (*C var*), 259
drivers_Kinetis_K_MK22F51212._ftfx_cache_config.flashMemoryIndex (*C var*), 258
drivers_Kinetis_K_MK22F51212._ftfx_cache_ram_func_constants (*C enum*), 257
drivers_Kinetis_K_MK22F51212._ftfx_cache_ram_func_constants.kFTFx_CACHE_RamFuncMaxSizeInWords (*C enumerator*), 257
drivers_Kinetis_K_MK22F51212._ftfx_config (*C struct*), 276
drivers_Kinetis_K_MK22F51212._ftfx_config.eepromTotalSize (*C var*), 276
drivers_Kinetis_K_MK22F51212._ftfx_config.flexramBlockBase (*C var*), 276
drivers_Kinetis_K_MK22F51212._ftfx_config.flexramTotalSize (*C var*), 276
drivers_Kinetis_K_MK22F51212._ftfx_config.runCmdFuncAddr (*C var*), 276
drivers_Kinetis_K_MK22F51212._ftfx_driver_api_keys (*C enum*), 260
drivers_Kinetis_K_MK22F51212._ftfx_driver_api_keys.kFTFx_ApiEraseKey (*C enumerator*), 260
drivers_Kinetis_K_MK22F51212._ftfx_flexram_function_option (*C enum*), 271
drivers_Kinetis_K_MK22F51212._ftfx_flexram_function_option.kFTFx_FlexramFuncOptAvailableAsRam (*C enumerator*), 272
drivers_Kinetis_K_MK22F51212._ftfx_flexram_function_option.kFTFx_FlexramFuncOptAvailableForEeprom (*C enumerator*), 272
drivers_Kinetis_K_MK22F51212._ftfx_flexram_function_option.kFTFx_FlexramFuncOptAvailableForEepromQuick (*C enumerator*), 272
drivers_Kinetis_K_MK22F51212._ftfx_flexram_function_option.kFTFx_FlexramFuncOptEepromQuickWriteRecover (*C enumerator*), 272
drivers_Kinetis_K_MK22F51212._ftfx_flexram_function_option.kFTFx_FlexramFuncOptEepromQuickWriteStatus (*C enumerator*), 272
drivers_Kinetis_K_MK22F51212._ftfx_ifr_descriptor (*C struct*), 275
drivers_Kinetis_K_MK22F51212._ftfx_ifr_descriptor.feature (*C struct*), 276
drivers_Kinetis_K_MK22F51212._ftfx_ifr_descriptor.idxInfo (*C struct*), 277
drivers_Kinetis_K_MK22F51212._ftfx_ifr_descriptor.resRange (*C struct*), 276
drivers_Kinetis_K_MK22F51212._ftfx_ifr_descriptor.resRange.dflashIfrStart (*C var*), 277
drivers_Kinetis_K_MK22F51212._ftfx_ifr_descriptor.resRange.pflashIfrStart (*C var*), 277
drivers_Kinetis_K_MK22F51212._ftfx_ifr_descriptor.resRange.pflashSwapIfrStart (*C var*), 277
drivers_Kinetis_K_MK22F51212._ftfx_ifr_descriptor.resRange.versionIdStart (*C var*), 277
drivers_Kinetis_K_MK22F51212._ftfx_margin_value (*C enum*), 271
drivers_Kinetis_K_MK22F51212._ftfx_margin_value.kFTFx_MarginValueFactory (*C enumerator*), 271
drivers_Kinetis_K_MK22F51212._ftfx_margin_value.kFTFx_MarginValueInvalid (*C enumerator*), 271
drivers_Kinetis_K_MK22F51212._ftfx_margin_value.kFTFx_MarginValueNormal (*C enumerator*), 271
drivers_Kinetis_K_MK22F51212._ftfx_margin_value.kFTFx_MarginValueUser (*C enumerator*), 271
drivers_Kinetis_K_MK22F51212._ftfx_mem_descriptor (*C struct*), 275
drivers_Kinetis_K_MK22F51212._ftfx_mem_descriptor.___unnamed11___ (*C struct*), 276
drivers_Kinetis_K_MK22F51212._ftfx_mem_descriptor.___unnamed11___.index (*C var*), 276
drivers_Kinetis_K_MK22F51212._ftfx_mem_descriptor.___unnamed11___.type (*C var*), 276
drivers_Kinetis_K_MK22F51212._ftfx_mem_descriptor.aliasBlockBase (*C var*), 275
drivers_Kinetis_K_MK22F51212._ftfx_mem_descriptor.blockBase (*C var*), 275
drivers_Kinetis_K_MK22F51212._ftfx_mem_descriptor.blockCount (*C var*), 275
drivers_Kinetis_K_MK22F51212._ftfx_mem_descriptor.feature (*C struct*), 276
drivers_Kinetis_K_MK22F51212._ftfx_mem_descriptor.sectorSize (*C var*), 275
drivers_Kinetis_K_MK22F51212._ftfx_mem_descriptor.totalSize (*C var*), 275
drivers_Kinetis_K_MK22F51212._ftfx_memory_type (*C enum*), 273
drivers_Kinetis_K_MK22F51212._ftfx_memory_type.kFTFx_MemTypeFlexnvm (*C enumerator*), 273
drivers_Kinetis_K_MK22F51212._ftfx_memory_type.kFTFx_MemTypePflash (*C enumerator*), 273
drivers_Kinetis_K_MK22F51212._ftfx_ops_config (*C struct*), 275

drivers_Kinetis_K_MK22F51212._ftfx_ops_config.addrAligment (*C struct*), 276
drivers_Kinetis_K_MK22F51212._ftfx_ops_config.convertedAddress (*C var*), 275
drivers_Kinetis_K_MK22F51212._ftfx_partition_flexram_load_option (*C enum*), 270
drivers_Kinetis_K_MK22F51212._ftfx_partition_flexram_load_option.kFTFx_PartitionFlexramLoadOptLoadedWit (*C enumerator*), 271
drivers_Kinetis_K_MK22F51212._ftfx_partition_flexram_load_option.kFTFx_PartitionFlexramLoadOptNotLoaded (*C enumerator*), 271
drivers_Kinetis_K_MK22F51212._ftfx_read_resource_opt (*C enum*), 271
drivers_Kinetis_K_MK22F51212._ftfx_read_resource_opt.kFTFx_ResourceOptionFlashIfr (*C enumerator*), 271
drivers_Kinetis_K_MK22F51212._ftfx_read_resource_opt.kFTFx_ResourceOptionVersionId (*C enumerator*), 271
drivers_Kinetis_K_MK22F51212._ftfx_security_state (*C enum*), 271
drivers_Kinetis_K_MK22F51212._ftfx_security_state.kFTFx_SecurityStateBackdoorDisabled (*C enumerator*), 271
drivers_Kinetis_K_MK22F51212._ftfx_security_state.kFTFx_SecurityStateBackdoorEnabled (*C enumerator*), 271
drivers_Kinetis_K_MK22F51212._ftfx_security_state.kFTFx_SecurityStateNotSecure (*C enumerator*), 271
drivers_Kinetis_K_MK22F51212._ftfx_special_mem (*C struct*), 274
drivers_Kinetis_K_MK22F51212._ftfx_special_mem.base (*C var*), 275
drivers_Kinetis_K_MK22F51212._ftfx_special_mem.count (*C var*), 275
drivers_Kinetis_K_MK22F51212._ftfx_special_mem.size (*C var*), 275
drivers_Kinetis_K_MK22F51212._ftfx_swap_block_status (*C enum*), 273
drivers_Kinetis_K_MK22F51212._ftfx_swap_block_status.kFTFx_SwapBlockStatusLowerHalfProgramBlocksAtZero (*C enumerator*), 273
drivers_Kinetis_K_MK22F51212._ftfx_swap_block_status.kFTFx_SwapBlockStatusUpperHalfProgramBlocksAtZero (*C enumerator*), 273
drivers_Kinetis_K_MK22F51212._ftfx_swap_control_option (*C enum*), 272
drivers_Kinetis_K_MK22F51212._ftfx_swap_control_option.kFTFx_SwapControlOptionDisableSystem (*C enumerator*), 272
drivers_Kinetis_K_MK22F51212._ftfx_swap_control_option.kFTFx_SwapControlOptionIntializeSystem (*C enumerator*), 272
drivers_Kinetis_K_MK22F51212._ftfx_swap_control_option.kFTFx_SwapControlOptionReportStatus (*C enumerator*), 272
drivers_Kinetis_K_MK22F51212._ftfx_swap_control_option.kFTFx_SwapControlOptionSetInCompleteState (*C enumerator*), 272
drivers_Kinetis_K_MK22F51212._ftfx_swap_control_option.kFTFx_SwapControlOptionSetInUpdateState (*C enumerator*), 272
drivers_Kinetis_K_MK22F51212._ftfx_swap_state (*C enum*), 272
drivers_Kinetis_K_MK22F51212._ftfx_swap_state_config (*C struct*), 274
drivers_Kinetis_K_MK22F51212._ftfx_swap_state_config.currentSwapBlockStatus (*C var*), 274
drivers_Kinetis_K_MK22F51212._ftfx_swap_state_config.flashSwapState (*C var*), 274
drivers_Kinetis_K_MK22F51212._ftfx_swap_state_config.nextSwapBlockStatus (*C var*), 274
drivers_Kinetis_K_MK22F51212._ftfx_swap_state.kFTFx_SwapStateComplete (*C enumerator*), 273
drivers_Kinetis_K_MK22F51212._ftfx_swap_state.kFTFx_SwapStateDisabled (*C enumerator*), 273
drivers_Kinetis_K_MK22F51212._ftfx_swap_state.kFTFx_SwapStateReady (*C enumerator*), 273
drivers_Kinetis_K_MK22F51212._ftfx_swap_state.kFTFx_SwapStateUninitialized (*C enumerator*), 272
drivers_Kinetis_K_MK22F51212._ftfx_swap_state.kFTFx_SwapStateUpdate (*C enumerator*), 273
drivers_Kinetis_K_MK22F51212._ftfx_swap_state.kFTFx_SwapStateUpdateErased (*C enumerator*), 273
drivers_Kinetis_K_MK22F51212._ftm_bdm_mode (*C enum*), 316
drivers_Kinetis_K_MK22F51212._ftm_bdm_mode.kFTM_BdmMode_0 (*C enumerator*), 316
drivers_Kinetis_K_MK22F51212._ftm_bdm_mode.kFTM_BdmMode_1 (*C enumerator*), 316
drivers_Kinetis_K_MK22F51212._ftm_bdm_mode.kFTM_BdmMode_2 (*C enumerator*), 317
drivers_Kinetis_K_MK22F51212._ftm_bdm_mode.kFTM_BdmMode_3 (*C enumerator*), 317

drivers_Kinetis_K_MK22F51212._ftm_channel_index (*C enum*), 321
drivers_Kinetis_K_MK22F51212._ftm_channel_index.kFTM_Chnl0_Mask (*C enumerator*), 321
drivers_Kinetis_K_MK22F51212._ftm_channel_index.kFTM_Chnl1_Mask (*C enumerator*), 321
drivers_Kinetis_K_MK22F51212._ftm_channel_index.kFTM_Chnl2_Mask (*C enumerator*), 321
drivers_Kinetis_K_MK22F51212._ftm_channel_index.kFTM_Chnl3_Mask (*C enumerator*), 321
drivers_Kinetis_K_MK22F51212._ftm_channel_index.kFTM_Chnl4_Mask (*C enumerator*), 321
drivers_Kinetis_K_MK22F51212._ftm_channel_index.kFTM_Chnl5_Mask (*C enumerator*), 321
drivers_Kinetis_K_MK22F51212._ftm_channel_index.kFTM_Chnl6_Mask (*C enumerator*), 321
drivers_Kinetis_K_MK22F51212._ftm_channel_index.kFTM_Chnl7_Mask (*C enumerator*), 321
drivers_Kinetis_K_MK22F51212._ftm_chnl (*C enum*), 312
drivers_Kinetis_K_MK22F51212._ftm_chnl_param (*C struct*), 327
drivers_Kinetis_K_MK22F51212._ftm_chnl_param.chnlValue (*C var*), 327
drivers_Kinetis_K_MK22F51212._ftm_chnl_param.combinedChnlValue (*C var*), 327
drivers_Kinetis_K_MK22F51212._ftm_chnl_param.enableComplementary (*C var*), 327
drivers_Kinetis_K_MK22F51212._ftm_chnl_param.enableDeadtime (*C var*), 327
drivers_Kinetis_K_MK22F51212._ftm_chnl_param.initialValue (*C var*), 327
drivers_Kinetis_K_MK22F51212._ftm_chnl_param.level (*C var*), 327
drivers_Kinetis_K_MK22F51212._ftm_chnl_param.mode (*C var*), 327
drivers_Kinetis_K_MK22F51212._ftm_chnl_param.moduloValue (*C var*), 327
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_config_param (*C struct*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_config_param.chnlNumber (*C var*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_config_param.dutyValue (*C var*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_config_param.enableComplementary (*C var*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_config_param.enableDeadtime (*C var*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_config_param.firstEdgeValue (*C var*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_config_param.level (*C var*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_signal_param (*C struct*), 325
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_signal_param.chnlNumber (*C var*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_signal_param.dutyCyclePercent (*C var*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_signal_param.enableComplementary (*C var*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_signal_param.enableDeadtime (*C var*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_signal_param.firstEdgeDelayPercent (*C var*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl_pwm_signal_param.level (*C var*), 326
drivers_Kinetis_K_MK22F51212._ftm_chnl.kFTM_Chnl_0 (*C enumerator*), 312
drivers_Kinetis_K_MK22F51212._ftm_chnl.kFTM_Chnl_1 (*C enumerator*), 312
drivers_Kinetis_K_MK22F51212._ftm_chnl.kFTM_Chnl_2 (*C enumerator*), 312
drivers_Kinetis_K_MK22F51212._ftm_chnl.kFTM_Chnl_3 (*C enumerator*), 312
drivers_Kinetis_K_MK22F51212._ftm_chnl.kFTM_Chnl_4 (*C enumerator*), 312
drivers_Kinetis_K_MK22F51212._ftm_chnl.kFTM_Chnl_5 (*C enumerator*), 313
drivers_Kinetis_K_MK22F51212._ftm_chnl.kFTM_Chnl_6 (*C enumerator*), 313
drivers_Kinetis_K_MK22F51212._ftm_chnl.kFTM_Chnl_7 (*C enumerator*), 313
drivers_Kinetis_K_MK22F51212._ftm_clock_prescale (*C enum*), 316
drivers_Kinetis_K_MK22F51212._ftm_clock_prescale.kFTM_Prescale_Divide_1 (*C enumerator*), 316
drivers_Kinetis_K_MK22F51212._ftm_clock_prescale.kFTM_Prescale_Divide_2 (*C enumerator*), 316
drivers_Kinetis_K_MK22F51212._ftm_clock_prescale.kFTM_Prescale_Divide_4 (*C enumerator*), 316
drivers_Kinetis_K_MK22F51212._ftm_clock_prescale.kFTM_Prescale_Divide_8 (*C enumerator*), 316
drivers_Kinetis_K_MK22F51212._ftm_clock_prescale.kFTM_Prescale_Divide_16 (*C enumerator*), 316
drivers_Kinetis_K_MK22F51212._ftm_clock_prescale.kFTM_Prescale_Divide_32 (*C enumerator*), 316
drivers_Kinetis_K_MK22F51212._ftm_clock_prescale.kFTM_Prescale_Divide_64 (*C enumerator*), 316
drivers_Kinetis_K_MK22F51212._ftm_clock_prescale.kFTM_Prescale_Divide_128 (*C enumerator*), 316

drivers_Kinetis_K_MK22F51212._ftm_clock_source (*C enum*), 315
drivers_Kinetis_K_MK22F51212._ftm_clock_source.kFTM_ExternalClock (*C enumerator*), 316
drivers_Kinetis_K_MK22F51212._ftm_clock_source.kFTM_FixedClock (*C enumerator*), 316
drivers_Kinetis_K_MK22F51212._ftm_clock_source.kFTM_SystemClock (*C enumerator*), 316
drivers_Kinetis_K_MK22F51212._ftm_config (*C struct*), 328
drivers_Kinetis_K_MK22F51212._ftm_config.bdmMode (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_config.chnlInitState (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_config.chnlPolarity (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_config.deadTimePrescale (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_config.deadTimeValue (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_config.extTriggers (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_config.faultFilterValue (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_config.faultMode (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_config.hwTriggerResetCount (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_config.prescale (*C var*), 328
drivers_Kinetis_K_MK22F51212._ftm_config.pwmSyncMode (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_config.reloadPoints (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_config.swTriggerResetCount (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_config.useGlobalTimeBase (*C var*), 329
drivers_Kinetis_K_MK22F51212._ftm_deadtime_prescale (*C enum*), 315
drivers_Kinetis_K_MK22F51212._ftm_deadtime_prescale.kFTM_Deadtime_Prescale_1 (*C enumerator*), 315
drivers_Kinetis_K_MK22F51212._ftm_deadtime_prescale.kFTM_Deadtime_Prescale_4 (*C enumerator*), 315
drivers_Kinetis_K_MK22F51212._ftm_deadtime_prescale.kFTM_Deadtime_Prescale_16 (*C enumerator*), 315
drivers_Kinetis_K_MK22F51212._ftm_dual_edge_capture_mode (*C enum*), 315
drivers_Kinetis_K_MK22F51212._ftm_dual_edge_capture_mode.kFTM_Continuous (*C enumerator*), 315
drivers_Kinetis_K_MK22F51212._ftm_dual_edge_capture_mode.kFTM_OneShot (*C enumerator*), 315
drivers_Kinetis_K_MK22F51212._ftm_dual_edge_capture_param (*C struct*), 327
drivers_Kinetis_K_MK22F51212._ftm_dual_edge_capture_param.currChanEdgeMode (*C var*), 327
drivers_Kinetis_K_MK22F51212._ftm_dual_edge_capture_param.mode (*C var*), 327
drivers_Kinetis_K_MK22F51212._ftm_dual_edge_capture_param.nextChanEdgeMode (*C var*), 328
drivers_Kinetis_K_MK22F51212._ftm_external_trigger (*C enum*), 317
drivers_Kinetis_K_MK22F51212._ftm_external_trigger.kFTM_Chnl0Trigger (*C enumerator*), 317
drivers_Kinetis_K_MK22F51212._ftm_external_trigger.kFTM_Chnl1Trigger (*C enumerator*), 317
drivers_Kinetis_K_MK22F51212._ftm_external_trigger.kFTM_Chnl2Trigger (*C enumerator*), 317
drivers_Kinetis_K_MK22F51212._ftm_external_trigger.kFTM_Chnl3Trigger (*C enumerator*), 317
drivers_Kinetis_K_MK22F51212._ftm_external_trigger.kFTM_Chnl4Trigger (*C enumerator*), 317
drivers_Kinetis_K_MK22F51212._ftm_external_trigger.kFTM_Chnl5Trigger (*C enumerator*), 318
drivers_Kinetis_K_MK22F51212._ftm_external_trigger.kFTM_Chnl6Trigger (*C enumerator*), 318
drivers_Kinetis_K_MK22F51212._ftm_external_trigger.kFTM_Chnl7Trigger (*C enumerator*), 318
drivers_Kinetis_K_MK22F51212._ftm_external_trigger.kFTM_InitTrigger (*C enumerator*), 318
drivers_Kinetis_K_MK22F51212._ftm_external_trigger.kFTM_ReloadInitTrigger (*C enumerator*), 318
drivers_Kinetis_K_MK22F51212._ftm_fault_input (*C enum*), 313
drivers_Kinetis_K_MK22F51212._ftm_fault_input.kFTM_Fault_0 (*C enumerator*), 313
drivers_Kinetis_K_MK22F51212._ftm_fault_input.kFTM_Fault_1 (*C enumerator*), 313
drivers_Kinetis_K_MK22F51212._ftm_fault_input.kFTM_Fault_2 (*C enumerator*), 313
drivers_Kinetis_K_MK22F51212._ftm_fault_input.kFTM_Fault_3 (*C enumerator*), 313
drivers_Kinetis_K_MK22F51212._ftm_fault_mode (*C enum*), 317
drivers_Kinetis_K_MK22F51212._ftm_fault_mode.kFTM_Fault_AllChnlsAuto (*C enumerator*), 317
drivers_Kinetis_K_MK22F51212._ftm_fault_mode.kFTM_Fault_AllChnlsMan (*C enumerator*), 317
drivers_Kinetis_K_MK22F51212._ftm_fault_mode.kFTM_Fault_Disable (*C enumerator*), 317

drivers_Kinetis_K_MK22F51212._ftm_fault_mode.kFTM_Fault_EvenChnls (*C enumerator*), 317
drivers_Kinetis_K_MK22F51212._ftm_fault_param (*C struct*), 328
drivers_Kinetis_K_MK22F51212._ftm_fault_param.enableFaultInput (*C var*), 328
drivers_Kinetis_K_MK22F51212._ftm_fault_param.faultLevel (*C var*), 328
drivers_Kinetis_K_MK22F51212._ftm_fault_param.useFaultFilter (*C var*), 328
drivers_Kinetis_K_MK22F51212._ftm_input_capture_edge (*C enum*), 314
drivers_Kinetis_K_MK22F51212._ftm_input_capture_edge.kFTM_FallingEdge (*C enumerator*), 314
drivers_Kinetis_K_MK22F51212._ftm_input_capture_edge.kFTM_RiseAndFallEdge (*C enumerator*), 315
drivers_Kinetis_K_MK22F51212._ftm_input_capture_edge.kFTM_RisingEdge (*C enumerator*), 314
drivers_Kinetis_K_MK22F51212._ftm_interrupt_enable (*C enum*), 319
drivers_Kinetis_K_MK22F51212._ftm_interrupt_enable.kFTM_Chnl0InterruptEnable (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_interrupt_enable.kFTM_Chnl1InterruptEnable (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_interrupt_enable.kFTM_Chnl2InterruptEnable (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_interrupt_enable.kFTM_Chnl3InterruptEnable (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_interrupt_enable.kFTM_Chnl4InterruptEnable (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_interrupt_enable.kFTM_Chnl5InterruptEnable (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_interrupt_enable.kFTM_Chnl6InterruptEnable (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_interrupt_enable.kFTM_Chnl7InterruptEnable (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_interrupt_enable.kFTM_FaultInterruptEnable (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_interrupt_enable.kFTM_ReloadInterruptEnable (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_interrupt_enable.kFTM_TimeOverflowInterruptEnable (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_output_compare_mode (*C enum*), 314
drivers_Kinetis_K_MK22F51212._ftm_output_compare_mode.kFTM_ClearOnMatch (*C enumerator*), 314
drivers_Kinetis_K_MK22F51212._ftm_output_compare_mode.kFTM_NoOutputSignal (*C enumerator*), 314
drivers_Kinetis_K_MK22F51212._ftm_output_compare_mode.kFTM_SetOnMatch (*C enumerator*), 314
drivers_Kinetis_K_MK22F51212._ftm_output_compare_mode.kFTM_ToggleOnMatch (*C enumerator*), 314
drivers_Kinetis_K_MK22F51212._ftm_phase_param (*C struct*), 328
drivers_Kinetis_K_MK22F51212._ftm_phase_param.enablePhaseFilter (*C var*), 328
drivers_Kinetis_K_MK22F51212._ftm_phase_param.phaseFilterVal (*C var*), 328
drivers_Kinetis_K_MK22F51212._ftm_phase_param.phasePolarity (*C var*), 328
drivers_Kinetis_K_MK22F51212._ftm_phase_polarity (*C enum*), 315
drivers_Kinetis_K_MK22F51212._ftm_phase_polarity.kFTM_QuadPhaseInvert (*C enumerator*), 315
drivers_Kinetis_K_MK22F51212._ftm_phase_polarity.kFTM_QuadPhaseNormal (*C enumerator*), 315
drivers_Kinetis_K_MK22F51212._ftm_pwm_level_select (*C enum*), 313
drivers_Kinetis_K_MK22F51212._ftm_pwm_level_select.kFTM_HighTrue (*C enumerator*), 314
drivers_Kinetis_K_MK22F51212._ftm_pwm_level_select.kFTM_LowTrue (*C enumerator*), 314
drivers_Kinetis_K_MK22F51212._ftm_pwm_level_select.kFTM_NoPwmSignal (*C enumerator*), 314

drivers_Kinetis_K_MK22F51212._ftm_pwm_mode (*C enum*), 313
drivers_Kinetis_K_MK22F51212._ftm_pwm_mode.kFTM_AsymmetricalCombinedPwm (*C enumerator*), 313
drivers_Kinetis_K_MK22F51212._ftm_pwm_mode.kFTM_CenterAlignedCombinedPwm (*C enumerator*), 313
drivers_Kinetis_K_MK22F51212._ftm_pwm_mode.kFTM_CenterAlignedPwm (*C enumerator*), 313
drivers_Kinetis_K_MK22F51212._ftm_pwm_mode.kFTM_EdgeAlignedCombinedPwm (*C enumerator*), 313
drivers_Kinetis_K_MK22F51212._ftm_pwm_mode.kFTM_EdgeAlignedPwm (*C enumerator*), 313
drivers_Kinetis_K_MK22F51212._ftm_pwm_sync_method (*C enum*), 318
drivers_Kinetis_K_MK22F51212._ftm_pwm_sync_method.kFTM_HardwareTrigger_0 (*C enumerator*), 318
drivers_Kinetis_K_MK22F51212._ftm_pwm_sync_method.kFTM_HardwareTrigger_1 (*C enumerator*), 318
drivers_Kinetis_K_MK22F51212._ftm_pwm_sync_method.kFTM_HardwareTrigger_2 (*C enumerator*), 318
drivers_Kinetis_K_MK22F51212._ftm_pwm_sync_method.kFTM_SoftwareTrigger (*C enumerator*), 318
drivers_Kinetis_K_MK22F51212._ftm_quad_decode_mode (*C enum*), 315
drivers_Kinetis_K_MK22F51212._ftm_quad_decode_mode.kFTM_QuadCountAndDir (*C enumerator*), 315
drivers_Kinetis_K_MK22F51212._ftm_quad_decode_mode.kFTM_QuadPhaseEncode (*C enumerator*), 315
drivers_Kinetis_K_MK22F51212._ftm_reload_point (*C enum*), 318
drivers_Kinetis_K_MK22F51212._ftm_reload_point.kFTM_Chnl0Match (*C enumerator*), 318
drivers_Kinetis_K_MK22F51212._ftm_reload_point.kFTM_Chnl1Match (*C enumerator*), 318
drivers_Kinetis_K_MK22F51212._ftm_reload_point.kFTM_Chnl2Match (*C enumerator*), 318
drivers_Kinetis_K_MK22F51212._ftm_reload_point.kFTM_Chnl3Match (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_reload_point.kFTM_Chnl4Match (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_reload_point.kFTM_Chnl5Match (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_reload_point.kFTM_Chnl6Match (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_reload_point.kFTM_Chnl7Match (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_reload_point.kFTM_CntMax (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_reload_point.kFTM_CntMin (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_reload_point.kFTM_HalfCycMatch (*C enumerator*), 319
drivers_Kinetis_K_MK22F51212._ftm_status_flags (*C enum*), 320
drivers_Kinetis_K_MK22F51212._ftm_status_flags.kFTM_Chnl0Flag (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_status_flags.kFTM_Chnl1Flag (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_status_flags.kFTM_Chnl2Flag (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_status_flags.kFTM_Chnl3Flag (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_status_flags.kFTM_Chnl4Flag (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_status_flags.kFTM_Chnl5Flag (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_status_flags.kFTM_Chnl6Flag (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_status_flags.kFTM_Chnl7Flag (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_status_flags.kFTM_ChnlTriggerFlag (*C enumerator*), 321
drivers_Kinetis_K_MK22F51212._ftm_status_flags.kFTM_FaultFlag (*C enumerator*), 320
drivers_Kinetis_K_MK22F51212._ftm_status_flags.kFTM_ReloadFlag (*C enumerator*), 321
drivers_Kinetis_K_MK22F51212._ftm_status_flags.kFTM_TimeOverflowFlag (*C enumerator*), 321
drivers_Kinetis_K_MK22F51212._gpio_checker_attribute (*C enum*), 330
drivers_Kinetis_K_MK22F51212._gpio_checker_attribute.kGPIO_IgnoreAttributeCheck (*C enumerator*), 330
drivers_Kinetis_K_MK22F51212._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureNPrivilegedsecureN (*C enumerator*), 330
drivers_Kinetis_K_MK22F51212._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureNPrivilegedsecureR (*C enumerator*), 330
drivers_Kinetis_K_MK22F51212._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureNPrivilegedsecureRW (*C enumerator*), 330

drivers_Kinetis_K_MK22F51212._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureRPrivilegedsecureRW (*C enumerator*), 330
drivers_Kinetis_K_MK22F51212._gpio_checker_attribute.kGPIO_UsernonsecureNUsersecureRWPrivilegedsecureRW (*C enumerator*), 330
drivers_Kinetis_K_MK22F51212._gpio_checker_attribute.kGPIO_UsernonsecureRUsersecureRPrivilegedsecureRW (*C enumerator*), 330
drivers_Kinetis_K_MK22F51212._gpio_checker_attribute.kGPIO_UsernonsecureRUsersecureRWPrivilegedsecureRW (*C enumerator*), 330
drivers_Kinetis_K_MK22F51212._gpio_checker_attribute.kGPIO_UsernonsecureRWUsersecureRWPrivilegedsecureR (*C enumerator*), 330
drivers_Kinetis_K_MK22F51212._gpio_pin_config (*C struct*), 331
drivers_Kinetis_K_MK22F51212._gpio_pin_config.outputLogic (*C var*), 331
drivers_Kinetis_K_MK22F51212._gpio_pin_config.pinDirection (*C var*), 331
drivers_Kinetis_K_MK22F51212._gpio_pin_direction (*C enum*), 330
drivers_Kinetis_K_MK22F51212._gpio_pin_direction.kGPIO_DigitalInput (*C enumerator*), 330
drivers_Kinetis_K_MK22F51212._gpio_pin_direction.kGPIO_DigitalOutput (*C enumerator*), 330
drivers_Kinetis_K_MK22F51212._i2c_direction (*C enum*), 344
drivers_Kinetis_K_MK22F51212._i2c_direction.kI2C_Read (*C enumerator*), 344
drivers_Kinetis_K_MK22F51212._i2c_direction.kI2C_Write (*C enumerator*), 344
drivers_Kinetis_K_MK22F51212._i2c_flags (*C enum*), 342
drivers_Kinetis_K_MK22F51212._i2c_flags.kI2C_AddressMatchFlag (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_flags.kI2C_ArbitrationLostFlag (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_flags.kI2C_BusBusyFlag (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_flags.kI2C_IntPendingFlag (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_flags.kI2C_RangeAddressMatchFlag (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_flags.kI2C_ReceiveNakFlag (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_flags.kI2C_StartDetectFlag (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_flags.kI2C_StopDetectFlag (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_flags.kI2C_TransferCompleteFlag (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_flags.kI2C_TransferDirectionFlag (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_interrupt_enable (*C enum*), 343
drivers_Kinetis_K_MK22F51212._i2c_interrupt_enable.kI2C_GlobalInterruptEnable (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_interrupt_enable.kI2C_StartStopDetectInterruptEnable (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_interrupt_enable.kI2C_StopDetectInterruptEnable (*C enumerator*), 343
drivers_Kinetis_K_MK22F51212._i2c_master_config (*C struct*), 346
drivers_Kinetis_K_MK22F51212._i2c_master_config.baudRate_Bps (*C var*), 347
drivers_Kinetis_K_MK22F51212._i2c_master_config.enableDoubleBuffering (*C var*), 346
drivers_Kinetis_K_MK22F51212._i2c_master_config.enableMaster (*C var*), 346
drivers_Kinetis_K_MK22F51212._i2c_master_config.enableStopHold (*C var*), 346
drivers_Kinetis_K_MK22F51212._i2c_master_config.glitchFilterWidth (*C var*), 347
drivers_Kinetis_K_MK22F51212._i2c_master_edma_handle (*C struct*), 350
drivers_Kinetis_K_MK22F51212._i2c_master_edma_handle.completionCallback (*C var*), 351
drivers_Kinetis_K_MK22F51212._i2c_master_edma_handle.dmaHandle (*C var*), 351
drivers_Kinetis_K_MK22F51212._i2c_master_edma_handle.nbytes (*C var*), 351
drivers_Kinetis_K_MK22F51212._i2c_master_edma_handle.state (*C var*), 351
drivers_Kinetis_K_MK22F51212._i2c_master_edma_handle.transfer (*C var*), 351
drivers_Kinetis_K_MK22F51212._i2c_master_edma_handle.transferSize (*C var*), 351
drivers_Kinetis_K_MK22F51212._i2c_master_edma_handle.userData (*C var*), 351
drivers_Kinetis_K_MK22F51212._i2c_master_handle (*C struct*), 348
drivers_Kinetis_K_MK22F51212._i2c_master_handle.completionCallback (*C var*), 348
drivers_Kinetis_K_MK22F51212._i2c_master_handle.state (*C var*), 348
drivers_Kinetis_K_MK22F51212._i2c_master_handle.transfer (*C var*), 348
drivers_Kinetis_K_MK22F51212._i2c_master_handle.transferSize (*C var*), 348
drivers_Kinetis_K_MK22F51212._i2c_master_handle.userData (*C var*), 348
drivers_Kinetis_K_MK22F51212._i2c_master_transfer (*C struct*), 347

drivers_Kinetis_K_MK22F51212._i2c_master_transfer_flags (*C enum*), 344
drivers_Kinetis_K_MK22F51212._i2c_master_transfer_flags.kI2C_TransferDefaultFlag (*C enumerator*), 344
drivers_Kinetis_K_MK22F51212._i2c_master_transfer_flags.kI2C_TransferNoStartFlag (*C enumerator*), 344
drivers_Kinetis_K_MK22F51212._i2c_master_transfer_flags.kI2C_TransferNoStopFlag (*C enumerator*), 344
drivers_Kinetis_K_MK22F51212._i2c_master_transfer_flags.kI2C_TransferRepeatedStartFlag (*C enumerator*), 344
drivers_Kinetis_K_MK22F51212._i2c_master_transfer.data (*C var*), 348
drivers_Kinetis_K_MK22F51212._i2c_master_transfer.dataSize (*C var*), 348
drivers_Kinetis_K_MK22F51212._i2c_master_transfer.direction (*C var*), 348
drivers_Kinetis_K_MK22F51212._i2c_master_transfer.flags (*C var*), 348
drivers_Kinetis_K_MK22F51212._i2c_master_transfer.slaveAddress (*C var*), 348
drivers_Kinetis_K_MK22F51212._i2c_master_transfer.subaddress (*C var*), 348
drivers_Kinetis_K_MK22F51212._i2c_master_transfer.subaddressSize (*C var*), 348
drivers_Kinetis_K_MK22F51212._i2c_slave_address_mode (*C enum*), 344
drivers_Kinetis_K_MK22F51212._i2c_slave_address_mode.kI2C_Address7bit (*C enumerator*), 344
drivers_Kinetis_K_MK22F51212._i2c_slave_address_mode.kI2C_RangeMatch (*C enumerator*), 344
drivers_Kinetis_K_MK22F51212._i2c_slave_config (*C struct*), 347
drivers_Kinetis_K_MK22F51212._i2c_slave_config.addressingMode (*C var*), 347
drivers_Kinetis_K_MK22F51212._i2c_slave_config.enableBaudRateCtl (*C var*), 347
drivers_Kinetis_K_MK22F51212._i2c_slave_config.enableDoubleBuffering (*C var*), 347
drivers_Kinetis_K_MK22F51212._i2c_slave_config.enableGeneralCall (*C var*), 347
drivers_Kinetis_K_MK22F51212._i2c_slave_config.enableSlave (*C var*), 347
drivers_Kinetis_K_MK22F51212._i2c_slave_config.enableWakeUp (*C var*), 347
drivers_Kinetis_K_MK22F51212._i2c_slave_config.sclStopHoldTime_ns (*C var*), 347
drivers_Kinetis_K_MK22F51212._i2c_slave_config.slaveAddress (*C var*), 347
drivers_Kinetis_K_MK22F51212._i2c_slave_config.upperAddress (*C var*), 347
drivers_Kinetis_K_MK22F51212._i2c_slave_handle (*C struct*), 349
drivers_Kinetis_K_MK22F51212._i2c_slave_handle.callback (*C var*), 349
drivers_Kinetis_K_MK22F51212._i2c_slave_handle.eventMask (*C var*), 349
drivers_Kinetis_K_MK22F51212._i2c_slave_handle.isBusy (*C var*), 349
drivers_Kinetis_K_MK22F51212._i2c_slave_handle.transfer (*C var*), 349
drivers_Kinetis_K_MK22F51212._i2c_slave_handle.userData (*C var*), 349
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer (*C struct*), 348
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer_event (*C enum*), 344
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer_event.kI2C_SlaveAddressMatchEvent (*C enumerator*), 345
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer_event.kI2C_SlaveAllEvents (*C enumerator*), 345
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer_event.kI2C_SlaveCompletionEvent (*C enumerator*), 345
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer_event.kI2C_SlaveGenaralcallEvent (*C enumerator*), 345
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer_event.kI2C_SlaveReceiveEvent (*C enumerator*), 345
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer_event.kI2C_SlaveStartEvent (*C enumerator*), 345
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer_event.kI2C_SlaveTransmitAckEvent (*C enumerator*), 345
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer_event.kI2C_SlaveTransmitEvent (*C enumerator*), 345
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer.completionStatus (*C var*), 349
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer.data (*C var*), 349
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer.dataSize (*C var*), 349
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer.event (*C var*), 349
drivers_Kinetis_K_MK22F51212._i2c_slave_transfer.transferredCount (*C var*), 349

drivers_Kinetis_K_MK22F51212._lin_lpuart_flags (*C enum*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_DataMatch1Flag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_DataMatch2Flag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_FramingErrorFlag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_IdleLineFlag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_LinBreakFlag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_NoiseErrorFlag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_NoiseErrorInRxDataRegFlag (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_ParityErrorFlag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_ParityErrorInRxDataRegFlag (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_RxActiveEdgeFlag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_RxActiveFlag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_RxDataRegFullFlag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_RxFifoEmptyFlag (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_RxFifoUnderflowFlag (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_RxOverrunFlag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_TransmissionCompleteFlag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_TxDataRegEmptyFlag (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_TxFifoEmptyFlag (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_flags.kLPUART_TxFifoOverflowFlag (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable (*C enum*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable.kLPUART_FramingErrorInterruptEnable (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable.kLPUART_IdleLineInterruptEnable (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable.kLPUART_LinBreakInterruptEnable (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable.kLPUART_NoiseErrorInterruptEnable (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable.kLPUART_ParityErrorInterruptEnable (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable.kLPUART_RxActiveEdgeInterruptEnable (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable.kLPUART_RxDataRegFullInterruptEnable (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable.kLPUART_RxFifoUnderflowInterruptEnable (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable.kLPUART_RxOverrunInterruptEnable (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable.kLPUART_TransmissionCompleteInterruptEnable (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable.kLPUART_TxDataRegEmptyInterruptEnable (*C enumerator*), 369
drivers_Kinetis_K_MK22F51212._lin_lpuart_interrupt_enable.kLPUART_TxFifoOverflowInterruptEnable (*C enumerator*), 370

drivers_Kinetis_K_MK22F51212._lin_lpuart_status (*C enum*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_Error (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_FlagCannotClearManually (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_FramingError (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_NoiseError (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_ParityError (*C enumerator*), 371
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_RxBusy (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_RxHardwareOverrun (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_RxIdle (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_RxRingBufferOverrun (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_RxWatermarkTooLarge (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_TxBusy (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_TxIdle (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_status.kStatus_LPUART_TxWatermarkTooLarge (*C enumerator*), 370
drivers_Kinetis_K_MK22F51212._lin_lpuart_stop_bit_count (*C enum*), 367
drivers_Kinetis_K_MK22F51212._lin_lpuart_stop_bit_count.kLPUART_OneStopBit (*C enumerator*), 367
drivers_Kinetis_K_MK22F51212._lin_lpuart_stop_bit_count.kLPUART_TwoStopBit (*C enumerator*), 368
drivers_Kinetis_K_MK22F51212._llwu_external_pin_filter_mode (*C struct*), 380
drivers_Kinetis_K_MK22F51212._llwu_external_pin_filter_mode.filterMode (*C var*), 380
drivers_Kinetis_K_MK22F51212._llwu_external_pin_filter_mode.pinIndex (*C var*), 380
drivers_Kinetis_K_MK22F51212._llwu_external_pin_mode (*C enum*), 378
drivers_Kinetis_K_MK22F51212._llwu_external_pin_mode.kLLWU_ExternalPinAnyEdge (*C enumerator*), 379
drivers_Kinetis_K_MK22F51212._llwu_external_pin_mode.kLLWU_ExternalPinDisable (*C enumerator*), 378
drivers_Kinetis_K_MK22F51212._llwu_external_pin_mode.kLLWU_ExternalPinFallingEdge (*C enumerator*), 379
drivers_Kinetis_K_MK22F51212._llwu_external_pin_mode.kLLWU_ExternalPinRisingEdge (*C enumerator*), 379
drivers_Kinetis_K_MK22F51212._llwu_param (*C struct*), 380
drivers_Kinetis_K_MK22F51212._llwu_param.dmas (*C var*), 380
drivers_Kinetis_K_MK22F51212._llwu_param.filters (*C var*), 380
drivers_Kinetis_K_MK22F51212._llwu_param.modules (*C var*), 380
drivers_Kinetis_K_MK22F51212._llwu_param.pins (*C var*), 380
drivers_Kinetis_K_MK22F51212._llwu_pin_filter_mode (*C enum*), 379
drivers_Kinetis_K_MK22F51212._llwu_pin_filter_mode.kLLWU_PinFilterAnyEdge (*C enumerator*), 379
drivers_Kinetis_K_MK22F51212._llwu_pin_filter_mode.kLLWU_PinFilterDisable (*C enumerator*), 379
drivers_Kinetis_K_MK22F51212._llwu_pin_filter_mode.kLLWU_PinFilterFallingEdge (*C enumerator*), 379
drivers_Kinetis_K_MK22F51212._llwu_pin_filter_mode.kLLWU_PinFilterRisingEdge (*C enumerator*), 379
drivers_Kinetis_K_MK22F51212._llwu_version_id (*C struct*), 379
drivers_Kinetis_K_MK22F51212._llwu_version_id.feature (*C var*), 380

drivers_Kinetis_K_MK22F51212._llwu_version_id.major (*C var*), 380
drivers_Kinetis_K_MK22F51212._llwu_version_id.minor (*C var*), 380
drivers_Kinetis_K_MK22F51212._lptmr_config (*C struct*), 386
drivers_Kinetis_K_MK22F51212._lptmr_config.bypassPrescaler (*C var*), 387
drivers_Kinetis_K_MK22F51212._lptmr_config.enableFreeRunning (*C var*), 387
drivers_Kinetis_K_MK22F51212._lptmr_config.pinPolarity (*C var*), 386
drivers_Kinetis_K_MK22F51212._lptmr_config.pinSelect (*C var*), 386
drivers_Kinetis_K_MK22F51212._lptmr_config.prescalerClockSource (*C var*), 387
drivers_Kinetis_K_MK22F51212._lptmr_config.timerMode (*C var*), 386
drivers_Kinetis_K_MK22F51212._lptmr_config.value (*C var*), 387
drivers_Kinetis_K_MK22F51212._lptmr_interrupt_enable (*C enum*), 385
drivers_Kinetis_K_MK22F51212._lptmr_interrupt_enable.kLPTMR_TimerInterruptEnable (*C enumerator*), 385
drivers_Kinetis_K_MK22F51212._lptmr_pin_polarity (*C enum*), 383
drivers_Kinetis_K_MK22F51212._lptmr_pin_polarity.kLPTMR_PinPolarityActiveHigh (*C enumerator*), 383
drivers_Kinetis_K_MK22F51212._lptmr_pin_polarity.kLPTMR_PinPolarityActiveLow (*C enumerator*), 383
drivers_Kinetis_K_MK22F51212._lptmr_pin_select (*C enum*), 383
drivers_Kinetis_K_MK22F51212._lptmr_pin_select.kLPTMR_PinSelectInput_0 (*C enumerator*), 383
drivers_Kinetis_K_MK22F51212._lptmr_pin_select.kLPTMR_PinSelectInput_1 (*C enumerator*), 383
drivers_Kinetis_K_MK22F51212._lptmr_pin_select.kLPTMR_PinSelectInput_2 (*C enumerator*), 383
drivers_Kinetis_K_MK22F51212._lptmr_pin_select.kLPTMR_PinSelectInput_3 (*C enumerator*), 383
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_clock_select (*C enum*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_clock_select.kLPTMR_PrescalerClock_0 (*C enumerator*), 385
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_clock_select.kLPTMR_PrescalerClock_1 (*C enumerator*), 385
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_clock_select.kLPTMR_PrescalerClock_2 (*C enumerator*), 385
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_clock_select.kLPTMR_PrescalerClock_3 (*C enumerator*), 385
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value (*C enum*), 383
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_0 (*C enumerator*), 383
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_1 (*C enumerator*), 383
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_2 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_3 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_4 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_5 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_6 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_7 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_8 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_9 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_10 (*C*

enumerator), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_11 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_12 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_13 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_14 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_prescaler_glitch_value.kLPTMR_Prescale_Glitch_15 (*C enumerator*), 384
drivers_Kinetis_K_MK22F51212._lptmr_status_flags (*C enum*), 385
drivers_Kinetis_K_MK22F51212._lptmr_status_flags.kLPTMR_TimerCompareFlag (*C enumerator*), 385
drivers_Kinetis_K_MK22F51212._lptmr_timer_mode (*C enum*), 383
drivers_Kinetis_K_MK22F51212._lptmr_timer_mode.kLPTMR_TimerModePulseCounter (*C enumerator*), 383
drivers_Kinetis_K_MK22F51212._lptmr_timer_mode.kLPTMR_TimerModeTimeCounter (*C enumerator*), 383
drivers_Kinetis_K_MK22F51212._lpuart_config (*C struct*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.baudRate_Bps (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.dataBitsCount (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.enableRx (*C var*), 406
drivers_Kinetis_K_MK22F51212._lpuart_config.enableRxRTS (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.enableTx (*C var*), 406
drivers_Kinetis_K_MK22F51212._lpuart_config.enableTxCTS (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.enableTxRTS (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.inverseTxd (*C var*), 406
drivers_Kinetis_K_MK22F51212._lpuart_config.isMsb (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.parityMode (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.rxFifoWatermark (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.rxIdleConfig (*C var*), 406
drivers_Kinetis_K_MK22F51212._lpuart_config.rxIdleType (*C var*), 406
drivers_Kinetis_K_MK22F51212._lpuart_config.stopBitCount (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.txCtsConfig (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.txCtsSource (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.txFifoWatermark (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_config.txRtsPolarity (*C var*), 405
drivers_Kinetis_K_MK22F51212._lpuart_data_bits (*C enum*), 399
drivers_Kinetis_K_MK22F51212._lpuart_data_bits.kLPUART_EightDataBits (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212._lpuart_data_bits.kLPUART_SevenDataBits (*C enumerator*), 400
drivers_Kinetis_K_MK22F51212._lpuart_edma_handle (*C struct*), 411
drivers_Kinetis_K_MK22F51212._lpuart_edma_handle.callback (*C var*), 411
drivers_Kinetis_K_MK22F51212._lpuart_edma_handle.nbytes (*C var*), 411
drivers_Kinetis_K_MK22F51212._lpuart_edma_handle.rxDataSizeAll (*C var*), 411
drivers_Kinetis_K_MK22F51212._lpuart_edma_handle.rxEdmaHandle (*C var*), 411
drivers_Kinetis_K_MK22F51212._lpuart_edma_handle.rxState (*C var*), 411
drivers_Kinetis_K_MK22F51212._lpuart_edma_handle.txDataSizeAll (*C var*), 411
drivers_Kinetis_K_MK22F51212._lpuart_edma_handle.txEdmaHandle (*C var*), 411
drivers_Kinetis_K_MK22F51212._lpuart_edma_handle.txState (*C var*), 411
drivers_Kinetis_K_MK22F51212._lpuart_edma_handle.userData (*C var*), 411
drivers_Kinetis_K_MK22F51212._lpuart_flags (*C enum*), 402
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_AllClearFlags (*C enumerator*), 404
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_AllFlags (*C enumerator*), 404
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_DataMatch1Flag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_DataMatch2Flag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_FramingErrorFlag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_IdleLineFlag (*C enumerator*), 403

drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_LinBreakFlag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_NoiseErrorFlag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_ParityErrorFlag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_RxActiveEdgeFlag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_RxActiveFlag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_RxDataRegFullFlag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_RxFifoEmptyFlag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_RxFifoUnderflowFlag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_RxOverrunFlag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_TransmissionCompleteFlag (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_TxDataRegEmptyFlag (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_TxFifoEmptyFlag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_flags.kLPUART_TxFifoOverflowFlag (*C enumerator*), 403
drivers_Kinetis_K_MK22F51212._lpuart_handle (*C struct*), 406
drivers_Kinetis_K_MK22F51212._lpuart_handle.___unnamed23___ (*C union*), 407
drivers_Kinetis_K_MK22F51212._lpuart_handle.___unnamed23___.txData (*C var*), 408
drivers_Kinetis_K_MK22F51212._lpuart_handle.___unnamed23___.txData16 (*C var*), 408
drivers_Kinetis_K_MK22F51212._lpuart_handle.___unnamed25___ (*C union*), 408
drivers_Kinetis_K_MK22F51212._lpuart_handle.___unnamed25___.rxData (*C var*), 408
drivers_Kinetis_K_MK22F51212._lpuart_handle.___unnamed25___.rxData16 (*C var*), 408
drivers_Kinetis_K_MK22F51212._lpuart_handle.___unnamed27___ (*C union*), 408
drivers_Kinetis_K_MK22F51212._lpuart_handle.___unnamed27___.rxRingBuffer (*C var*), 408
drivers_Kinetis_K_MK22F51212._lpuart_handle.___unnamed27___.rxRingBuffer16 (*C var*), 408
drivers_Kinetis_K_MK22F51212._lpuart_handle.callback (*C var*), 407
drivers_Kinetis_K_MK22F51212._lpuart_handle.is16bitData (*C var*), 407
drivers_Kinetis_K_MK22F51212._lpuart_handle.isSevenDataBits (*C var*), 407
drivers_Kinetis_K_MK22F51212._lpuart_handle.rxDataSize (*C var*), 406
drivers_Kinetis_K_MK22F51212._lpuart_handle.rxDataSizeAll (*C var*), 406
drivers_Kinetis_K_MK22F51212._lpuart_handle.rxRingBufferHead (*C var*), 406
drivers_Kinetis_K_MK22F51212._lpuart_handle.rxRingBufferSize (*C var*), 406
drivers_Kinetis_K_MK22F51212._lpuart_handle.rxRingBufferTail (*C var*), 407
drivers_Kinetis_K_MK22F51212._lpuart_handle.rxState (*C var*), 407
drivers_Kinetis_K_MK22F51212._lpuart_handle.txDataSize (*C var*), 406
drivers_Kinetis_K_MK22F51212._lpuart_handle.txDataSizeAll (*C var*), 406
drivers_Kinetis_K_MK22F51212._lpuart_handle.txState (*C var*), 407
drivers_Kinetis_K_MK22F51212._lpuart_handle.userData (*C var*), 407
drivers_Kinetis_K_MK22F51212._lpuart_idle_config (*C enum*), 401
drivers_Kinetis_K_MK22F51212._lpuart_idle_config.kLPUART_IdleCharacter1 (*C enumerator*), 401
drivers_Kinetis_K_MK22F51212._lpuart_idle_config.kLPUART_IdleCharacter2 (*C enumerator*), 401
drivers_Kinetis_K_MK22F51212._lpuart_idle_config.kLPUART_IdleCharacter4 (*C enumerator*), 401
drivers_Kinetis_K_MK22F51212._lpuart_idle_config.kLPUART_IdleCharacter8 (*C enumerator*), 401
drivers_Kinetis_K_MK22F51212._lpuart_idle_config.kLPUART_IdleCharacter16 (*C enumerator*), 401
drivers_Kinetis_K_MK22F51212._lpuart_idle_config.kLPUART_IdleCharacter32 (*C enumerator*), 401
drivers_Kinetis_K_MK22F51212._lpuart_idle_config.kLPUART_IdleCharacter64 (*C enumerator*), 401
drivers_Kinetis_K_MK22F51212._lpuart_idle_config.kLPUART_IdleCharacter128 (*C enumerator*), 401
drivers_Kinetis_K_MK22F51212._lpuart_idle_type_select (*C enum*), 400

drivers_Kinetis_K_MK22F51212._lpuart_idle_type_select.kLPUART_IdleTypeStartBit (*C enumerator*), 401
drivers_Kinetis_K_MK22F51212._lpuart_idle_type_select.kLPUART_IdleTypeStopBit (*C enumerator*), 401
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable (*C enum*), 401
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_AllInterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_FramingErrorInterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_IdleLineInterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_LinBreakInterruptEnable (*C enumerator*), 401
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_Match1InterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_Match2InterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_NoiseErrorInterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_ParityErrorInterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_RxActiveEdgeInterruptEnable (*C enumerator*), 401
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_RxDataRegFullInterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_RxFifoUnderflowInterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_RxOverrunInterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_TransmissionCompleteInterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_TxDataRegEmptyInterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_interrupt_enable.kLPUART_TxFifoOverflowInterruptEnable (*C enumerator*), 402
drivers_Kinetis_K_MK22F51212._lpuart_parity_mode (*C enum*), 399
drivers_Kinetis_K_MK22F51212._lpuart_parity_mode.kLPUART_ParityDisabled (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212._lpuart_parity_mode.kLPUART_ParityEven (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212._lpuart_parity_mode.kLPUART_ParityOdd (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212._lpuart_stop_bit_count (*C enum*), 400
drivers_Kinetis_K_MK22F51212._lpuart_stop_bit_count.kLPUART_OneStopBit (*C enumerator*), 400
drivers_Kinetis_K_MK22F51212._lpuart_stop_bit_count.kLPUART_TwoStopBit (*C enumerator*), 400
drivers_Kinetis_K_MK22F51212._lpuart_transfer (*C struct*), 406
drivers_Kinetis_K_MK22F51212._lpuart_transfer.___unnamed21___ (*C union*), 407
drivers_Kinetis_K_MK22F51212._lpuart_transfer.___unnamed21___.data (*C var*), 407
drivers_Kinetis_K_MK22F51212._lpuart_transfer.___unnamed21___.rxData (*C var*), 407
drivers_Kinetis_K_MK22F51212._lpuart_transfer.___unnamed21___.rxData16 (*C var*), 407
drivers_Kinetis_K_MK22F51212._lpuart_transfer.___unnamed21___.txData (*C var*), 407
drivers_Kinetis_K_MK22F51212._lpuart_transfer.___unnamed21___.txData16 (*C var*), 407
drivers_Kinetis_K_MK22F51212._lpuart_transfer.dataSize (*C var*), 406
drivers_Kinetis_K_MK22F51212._lpuart_transmit_cts_config (*C enum*), 400
drivers_Kinetis_K_MK22F51212._lpuart_transmit_cts_config.kLPUART_CtsSampleAtIdle (*C enumerator*), 400
drivers_Kinetis_K_MK22F51212._lpuart_transmit_cts_config.kLPUART_CtsSampleAtStart (*C enumerator*), 400

drivers_Kinetis_K_MK22F51212._lpuart_transmit_cts_source (*C enum*), 400
drivers_Kinetis_K_MK22F51212._lpuart_transmit_cts_source.kLPUART_CtsSourceMatchResult (*C enumerator*), 400
drivers_Kinetis_K_MK22F51212._lpuart_transmit_cts_source.kLPUART_CtsSourcePin (*C enumerator*), 400
drivers_Kinetis_K_MK22F51212._lpuart_transmit_rts_polarity (*C enum*), 400
drivers_Kinetis_K_MK22F51212._lpuart_transmit_rts_polarity.kLPUART_RtsPolarityHigh (*C enumerator*), 400
drivers_Kinetis_K_MK22F51212._lpuart_transmit_rts_polarity.kLPUART_RtsPolarityLow (*C enumerator*), 400
drivers_Kinetis_K_MK22F51212._mcg_atm_select (*C enum*), 155
drivers_Kinetis_K_MK22F51212._mcg_atm_select.kMCG_AtmSel4m (*C enumerator*), 155
drivers_Kinetis_K_MK22F51212._mcg_atm_select.kMCG_AtmSel32k (*C enumerator*), 155
drivers_Kinetis_K_MK22F51212._mcg_clkout_src (*C enum*), 154
drivers_Kinetis_K_MK22F51212._mcg_clkout_src.kMCG_ClkOutSrcExternal (*C enumerator*), 154
drivers_Kinetis_K_MK22F51212._mcg_clkout_src.kMCG_ClkOutSrcInternal (*C enumerator*), 154
drivers_Kinetis_K_MK22F51212._mcg_clkout_src.kMCG_ClkOutSrcOut (*C enumerator*), 154
drivers_Kinetis_K_MK22F51212._mcg_config (*C struct*), 176
drivers_Kinetis_K_MK22F51212._mcg_dmx32 (*C enum*), 154
drivers_Kinetis_K_MK22F51212._mcg_dmx32.kMCG_Dmx32Default (*C enumerator*), 154
drivers_Kinetis_K_MK22F51212._mcg_dmx32.kMCG_Dmx32Fine (*C enumerator*), 154
drivers_Kinetis_K_MK22F51212._mcg_drs (*C enum*), 154
drivers_Kinetis_K_MK22F51212._mcg_drs.kMCG_DrsHigh (*C enumerator*), 154
drivers_Kinetis_K_MK22F51212._mcg_drs.kMCG_DrsLow (*C enumerator*), 154
drivers_Kinetis_K_MK22F51212._mcg_drs.kMCG_DrsMid (*C enumerator*), 154
drivers_Kinetis_K_MK22F51212._mcg_drs.kMCG_DrsMidHigh (*C enumerator*), 154
drivers_Kinetis_K_MK22F51212._mcg_fll_src (*C enum*), 153
drivers_Kinetis_K_MK22F51212._mcg_fll_src.kMCG_FllSrcExternal (*C enumerator*), 153
drivers_Kinetis_K_MK22F51212._mcg_fll_src.kMCG_FllSrcInternal (*C enumerator*), 153
drivers_Kinetis_K_MK22F51212._mcg_irc_mode (*C enum*), 153
drivers_Kinetis_K_MK22F51212._mcg_irc_mode.kMCG_IrcFast (*C enumerator*), 153
drivers_Kinetis_K_MK22F51212._mcg_irc_mode.kMCG_IrcSlow (*C enumerator*), 153
drivers_Kinetis_K_MK22F51212._mcg_mode (*C enum*), 157
drivers_Kinetis_K_MK22F51212._mcg_mode.kMCG_ModeBLPE (*C enumerator*), 157
drivers_Kinetis_K_MK22F51212._mcg_mode.kMCG_ModeBLPI (*C enumerator*), 157
drivers_Kinetis_K_MK22F51212._mcg_mode.kMCG_ModeError (*C enumerator*), 157
drivers_Kinetis_K_MK22F51212._mcg_mode.kMCG_ModeFBE (*C enumerator*), 157
drivers_Kinetis_K_MK22F51212._mcg_mode.kMCG_ModeFBI (*C enumerator*), 157
drivers_Kinetis_K_MK22F51212._mcg_mode.kMCG_ModeFEE (*C enumerator*), 157
drivers_Kinetis_K_MK22F51212._mcg_mode.kMCG_ModeFEI (*C enumerator*), 157
drivers_Kinetis_K_MK22F51212._mcg_mode.kMCG_ModePBE (*C enumerator*), 157
drivers_Kinetis_K_MK22F51212._mcg_mode.kMCG_ModePEE (*C enumerator*), 157
drivers_Kinetis_K_MK22F51212._mcg_monitor_mode (*C enum*), 155
drivers_Kinetis_K_MK22F51212._mcg_monitor_mode.kMCG_MonitorInt (*C enumerator*), 155
drivers_Kinetis_K_MK22F51212._mcg_monitor_mode.kMCG_MonitorNone (*C enumerator*), 155
drivers_Kinetis_K_MK22F51212._mcg_monitor_mode.kMCG_MonitorReset (*C enumerator*), 155
drivers_Kinetis_K_MK22F51212._mcg_oscsel (*C enum*), 155
drivers_Kinetis_K_MK22F51212._mcg_oscsel.kMCG_OscselIrc (*C enumerator*), 155
drivers_Kinetis_K_MK22F51212._mcg_oscsel.kMCG_OscselOsc (*C enumerator*), 155
drivers_Kinetis_K_MK22F51212._mcg_oscsel.kMCG_OscselRtc (*C enumerator*), 155
drivers_Kinetis_K_MK22F51212._mcg_pll_clk_select (*C enum*), 155
drivers_Kinetis_K_MK22F51212._mcg_pll_clk_select.kMCG_PllClkSelPll0 (*C enumerator*), 155
drivers_Kinetis_K_MK22F51212._mcg_pll_clk_select.kMCG_PllClkSelPll1 (*C enumerator*), 155
drivers_Kinetis_K_MK22F51212._mcg_pll_config (*C struct*), 176
drivers_Kinetis_K_MK22F51212._mcg_pll_ref_src (*C enum*), 154
drivers_Kinetis_K_MK22F51212._mcg_pll_ref_src.kMCG_PllRefOsc0 (*C enumerator*), 154
drivers_Kinetis_K_MK22F51212._mcg_pll_ref_src.kMCG_PllRefOsc1 (*C enumerator*), 154
drivers_Kinetis_K_MK22F51212._mcm_buffer_fault_attribute (*C union*), 415

drivers_Kinetis_K_MK22F51212._mcm_buffer_fault_attribute._mcm_buffer_fault_attribut (*C struct*), 416
drivers_Kinetis_K_MK22F51212._mcm_buffer_fault_attribute._mcm_buffer_fault_attribut.busErrorAccess (*C var*), 416
drivers_Kinetis_K_MK22F51212._mcm_buffer_fault_attribute._mcm_buffer_fault_attribut.busErrorDataAccessTy (*C var*), 416
drivers_Kinetis_K_MK22F51212._mcm_buffer_fault_attribute._mcm_buffer_fault_attribut.busErrorMasterID (*C var*), 416
drivers_Kinetis_K_MK22F51212._mcm_buffer_fault_attribute._mcm_buffer_fault_attribut.busErrorOverrun (*C var*), 416
drivers_Kinetis_K_MK22F51212._mcm_buffer_fault_attribute._mcm_buffer_fault_attribut.busErrorPrivilegeLevel (*C var*), 416
drivers_Kinetis_K_MK22F51212._mcm_buffer_fault_attribute._mcm_buffer_fault_attribut.busErrorSize (*C var*), 416
drivers_Kinetis_K_MK22F51212._mcm_buffer_fault_attribute.attribute (*C var*), 415
drivers_Kinetis_K_MK22F51212._mcm_buffer_fault_attribute.attribute_memory (*C var*), 415
drivers_Kinetis_K_MK22F51212._mcm_lmem_fault_attribute (*C union*), 416
drivers_Kinetis_K_MK22F51212._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut (*C struct*), 416
drivers_Kinetis_K_MK22F51212._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut.backdoorAccess (*C var*), 417
drivers_Kinetis_K_MK22F51212._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut.overrun (*C var*), 417
drivers_Kinetis_K_MK22F51212._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut.parityFaultMasterSize (*C var*), 416
drivers_Kinetis_K_MK22F51212._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut.parityFaultProtectionSi (*C var*), 416
drivers_Kinetis_K_MK22F51212._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut.parityFaultSyndrome (*C var*), 417
drivers_Kinetis_K_MK22F51212._mcm_lmem_fault_attribute._mcm_lmem_fault_attribut.parityFaultWrite (*C var*), 417
drivers_Kinetis_K_MK22F51212._mcm_lmem_fault_attribute.attribute (*C var*), 416
drivers_Kinetis_K_MK22F51212._mcm_lmem_fault_attribute.attribute_memory (*C var*), 416
drivers_Kinetis_K_MK22F51212._osc_cap_load (*C enum*), 153
drivers_Kinetis_K_MK22F51212._osc_cap_load.kOSC_Cap2P (*C enumerator*), 153
drivers_Kinetis_K_MK22F51212._osc_cap_load.kOSC_Cap4P (*C enumerator*), 153
drivers_Kinetis_K_MK22F51212._osc_cap_load.kOSC_Cap8P (*C enumerator*), 153
drivers_Kinetis_K_MK22F51212._osc_cap_load.kOSC_Cap16P (*C enumerator*), 153
drivers_Kinetis_K_MK22F51212._osc_config (*C struct*), 176
drivers_Kinetis_K_MK22F51212._osc_mode (*C enum*), 152
drivers_Kinetis_K_MK22F51212._osc_mode.kOSC_ModeExt (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._osc_mode.kOSC_ModeOscHighGain (*C enumerator*), 153
drivers_Kinetis_K_MK22F51212._osc_mode.kOSC_ModeOscLowPower (*C enumerator*), 152
drivers_Kinetis_K_MK22F51212._oscer_config (*C struct*), 176
drivers_Kinetis_K_MK22F51212._oscer_enable_mode (*C enum*), 153
drivers_Kinetis_K_MK22F51212._oscer_enable_mode.kOSC_ErClkEnable (*C enumerator*), 153
drivers_Kinetis_K_MK22F51212._oscer_enable_mode.kOSC_ErClkEnableInStop (*C enumerator*), 153
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger (*C enum*), 424
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger_config (*C struct*), 428
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger_config.enableBackToBackOperationMask (*C var*), 428
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger_config.enableOutputMask (*C var*), 428
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger_config.enablePreTriggerMask (*C var*), 428
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger_flags (*C enum*), 421
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger_flags.kPDB_ADCPreTriggerChannel0ErrorFlag (*C enumerator*), 421
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger_flags.kPDB_ADCPreTriggerChannel0Flag

(*C enumerator*), 421
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger_flags.kPDB_ADCPreTriggerChannel1ErrorFlag (*C enumerator*), 421
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger_flags.kPDB_ADCPreTriggerChannel1Flag (*C enumerator*), 421
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger.kPDB_ADCPreTrigger0 (*C enumerator*), 424
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger.kPDB_ADCPreTrigger1 (*C enumerator*), 425
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger.kPDB_ADCPreTrigger2 (*C enumerator*), 425
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger.kPDB_ADCPreTrigger3 (*C enumerator*), 425
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger.kPDB_ADCPreTrigger4 (*C enumerator*), 425
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger.kPDB_ADCPreTrigger5 (*C enumerator*), 425
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger.kPDB_ADCPreTrigger6 (*C enumerator*), 425
drivers_Kinetis_K_MK22F51212._pdb_adc_pretrigger.kPDB_ADCPreTrigger7 (*C enumerator*), 425
drivers_Kinetis_K_MK22F51212._pdb_adc_trigger_channel (*C enum*), 424
drivers_Kinetis_K_MK22F51212._pdb_adc_trigger_channel.kPDB_ADCTriggerChannel0 (*C enumerator*), 424
drivers_Kinetis_K_MK22F51212._pdb_adc_trigger_channel.kPDB_ADCTriggerChannel1 (*C enumerator*), 424
drivers_Kinetis_K_MK22F51212._pdb_adc_trigger_channel.kPDB_ADCTriggerChannel2 (*C enumerator*), 424
drivers_Kinetis_K_MK22F51212._pdb_adc_trigger_channel.kPDB_ADCTriggerChannel3 (*C enumerator*), 424
drivers_Kinetis_K_MK22F51212._pdb_config (*C struct*), 427
drivers_Kinetis_K_MK22F51212._pdb_config.dividerMultiplicationFactor (*C var*), 428
drivers_Kinetis_K_MK22F51212._pdb_config.enableContinuousMode (*C var*), 428
drivers_Kinetis_K_MK22F51212._pdb_config.loadValueMode (*C var*), 428
drivers_Kinetis_K_MK22F51212._pdb_config.prescalerDivider (*C var*), 428
drivers_Kinetis_K_MK22F51212._pdb_config.triggerInputSource (*C var*), 428
drivers_Kinetis_K_MK22F51212._pdb_dac_trigger_channel (*C enum*), 425
drivers_Kinetis_K_MK22F51212._pdb_dac_trigger_channel.kPDB_DACTriggerChannel0 (*C enumerator*), 425
drivers_Kinetis_K_MK22F51212._pdb_dac_trigger_channel.kPDB_DACTriggerChannel1 (*C enumerator*), 425
drivers_Kinetis_K_MK22F51212._pdb_dac_trigger_config (*C struct*), 428
drivers_Kinetis_K_MK22F51212._pdb_dac_trigger_config.enableExternalTriggerInput (*C var*), 428
drivers_Kinetis_K_MK22F51212._pdb_dac_trigger_config.enableIntervalTrigger (*C var*), 428
drivers_Kinetis_K_MK22F51212._pdb_divider_multiplication_factor (*C enum*), 422
drivers_Kinetis_K_MK22F51212._pdb_divider_multiplication_factor.kPDB_DividerMultiplicationFactor1 (*C enumerator*), 423
drivers_Kinetis_K_MK22F51212._pdb_divider_multiplication_factor.kPDB_DividerMultiplicationFactor10 (*C enumerator*), 423
drivers_Kinetis_K_MK22F51212._pdb_divider_multiplication_factor.kPDB_DividerMultiplicationFactor20 (*C enumerator*), 423
drivers_Kinetis_K_MK22F51212._pdb_divider_multiplication_factor.kPDB_DividerMultiplicationFactor40 (*C enumerator*), 423
drivers_Kinetis_K_MK22F51212._pdb_interrupt_enable (*C enum*), 421
drivers_Kinetis_K_MK22F51212._pdb_interrupt_enable.kPDB_DelayInterruptEnable (*C enumerator*), 421
drivers_Kinetis_K_MK22F51212._pdb_interrupt_enable.kPDB_SequenceErrorInterruptEnable (*C enumerator*), 421

drivers_Kinetis_K_MK22F51212._pdb_load_value_mode (*C enum*), 421
drivers_Kinetis_K_MK22F51212._pdb_load_value_mode.kPDB_LoadValueImmediately (*C enumerator*), 422
drivers_Kinetis_K_MK22F51212._pdb_load_value_mode.kPDB_LoadValueOnCounterOverflow (*C enumerator*), 422
drivers_Kinetis_K_MK22F51212._pdb_load_value_mode.kPDB_LoadValueOnCounterOverflowOrTriggerInput (*C enumerator*), 422
drivers_Kinetis_K_MK22F51212._pdb_load_value_mode.kPDB_LoadValueOnTriggerInput (*C enumerator*), 422
drivers_Kinetis_K_MK22F51212._pdb_prescaler_divider (*C enum*), 422
drivers_Kinetis_K_MK22F51212._pdb_prescaler_divider.kPDB_PrescalerDivider1 (*C enumerator*), 422
drivers_Kinetis_K_MK22F51212._pdb_prescaler_divider.kPDB_PrescalerDivider2 (*C enumerator*), 422
drivers_Kinetis_K_MK22F51212._pdb_prescaler_divider.kPDB_PrescalerDivider4 (*C enumerator*), 422
drivers_Kinetis_K_MK22F51212._pdb_prescaler_divider.kPDB_PrescalerDivider8 (*C enumerator*), 422
drivers_Kinetis_K_MK22F51212._pdb_prescaler_divider.kPDB_PrescalerDivider16 (*C enumerator*), 422
drivers_Kinetis_K_MK22F51212._pdb_prescaler_divider.kPDB_PrescalerDivider32 (*C enumerator*), 422
drivers_Kinetis_K_MK22F51212._pdb_prescaler_divider.kPDB_PrescalerDivider64 (*C enumerator*), 422
drivers_Kinetis_K_MK22F51212._pdb_prescaler_divider.kPDB_PrescalerDivider128 (*C enumerator*), 422
drivers_Kinetis_K_MK22F51212._pdb_pulse_out_channel_mask (*C enum*), 426
drivers_Kinetis_K_MK22F51212._pdb_pulse_out_channel_mask.kPDB_PulseOutChannel0Mask (*C enumerator*), 426
drivers_Kinetis_K_MK22F51212._pdb_pulse_out_channel_mask.kPDB_PulseOutChannel1Mask (*C enumerator*), 426
drivers_Kinetis_K_MK22F51212._pdb_pulse_out_channel_mask.kPDB_PulseOutChannel2Mask (*C enumerator*), 426
drivers_Kinetis_K_MK22F51212._pdb_pulse_out_channel_mask.kPDB_PulseOutChannel3Mask (*C enumerator*), 426
drivers_Kinetis_K_MK22F51212._pdb_pulse_out_trigger_channel (*C enum*), 425
drivers_Kinetis_K_MK22F51212._pdb_pulse_out_trigger_channel.kPDB_PulseOutTriggerChannel0 (*C enumerator*), 425
drivers_Kinetis_K_MK22F51212._pdb_pulse_out_trigger_channel.kPDB_PulseOutTriggerChannel1 (*C enumerator*), 425
drivers_Kinetis_K_MK22F51212._pdb_pulse_out_trigger_channel.kPDB_PulseOutTriggerChannel2 (*C enumerator*), 426
drivers_Kinetis_K_MK22F51212._pdb_pulse_out_trigger_channel.kPDB_PulseOutTriggerChannel3 (*C enumerator*), 426
drivers_Kinetis_K_MK22F51212._pdb_status_flags (*C enum*), 421
drivers_Kinetis_K_MK22F51212._pdb_status_flags.kPDB_DelayEventFlag (*C enumerator*), 421
drivers_Kinetis_K_MK22F51212._pdb_status_flags.kPDB_LoadOKFlag (*C enumerator*), 421
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source (*C enum*), 423
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput0 (*C enumerator*), 423
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput1 (*C enumerator*), 423
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput2 (*C enumerator*), 423
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput3 (*C enumerator*), 423
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput4 (*C enumerator*), 423

drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput5 (*C enumerator*), 423
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput6 (*C enumerator*), 423
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput7 (*C enumerator*), 423
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput8 (*C enumerator*), 423
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput9 (*C enumerator*), 424
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput10 (*C enumerator*), 424
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput11 (*C enumerator*), 424
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput12 (*C enumerator*), 424
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput13 (*C enumerator*), 424
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerInput14 (*C enumerator*), 424
drivers_Kinetis_K_MK22F51212._pdb_trigger_input_source.kPDB_TriggerSoftware (*C enumerator*), 424
drivers_Kinetis_K_MK22F51212._pflash_protection_status (*C union*), 291
drivers_Kinetis_K_MK22F51212._pflash_protection_status.proth (*C var*), 291
drivers_Kinetis_K_MK22F51212._pflash_protection_status.protl (*C var*), 291
drivers_Kinetis_K_MK22F51212._pflash_protection_status.protsh (*C var*), 291
drivers_Kinetis_K_MK22F51212._pflash_protection_status.protsl (*C var*), 291
drivers_Kinetis_K_MK22F51212._pflash_protection_status.reserved (*C var*), 291
drivers_Kinetis_K_MK22F51212._pit_chnl (*C enum*), 431
drivers_Kinetis_K_MK22F51212._pit_chnl.kPIT_Chnl_0 (*C enumerator*), 431
drivers_Kinetis_K_MK22F51212._pit_chnl.kPIT_Chnl_1 (*C enumerator*), 431
drivers_Kinetis_K_MK22F51212._pit_chnl.kPIT_Chnl_2 (*C enumerator*), 431
drivers_Kinetis_K_MK22F51212._pit_chnl.kPIT_Chnl_3 (*C enumerator*), 432
drivers_Kinetis_K_MK22F51212._pit_config (*C struct*), 433
drivers_Kinetis_K_MK22F51212._pit_config.enableRunInDebug (*C var*), 433
drivers_Kinetis_K_MK22F51212._pit_interrupt_enable (*C enum*), 432
drivers_Kinetis_K_MK22F51212._pit_interrupt_enable.kPIT_TimerInterruptEnable (*C enumerator*), 432
drivers_Kinetis_K_MK22F51212._pit_status_flags (*C enum*), 432
drivers_Kinetis_K_MK22F51212._pit_status_flags.kPIT_TimerFlag (*C enumerator*), 432
drivers_Kinetis_K_MK22F51212._pmc_bandgap_buffer_config (*C struct*), 439
drivers_Kinetis_K_MK22F51212._pmc_bandgap_buffer_config.drive (*C var*), 439
drivers_Kinetis_K_MK22F51212._pmc_bandgap_buffer_config.enable (*C var*), 439
drivers_Kinetis_K_MK22F51212._pmc_bandgap_buffer_config.enableInLowPowerMode (*C var*), 439
drivers_Kinetis_K_MK22F51212._pmc_bandgap_buffer_drive_select (*C enum*), 436
drivers_Kinetis_K_MK22F51212._pmc_bandgap_buffer_drive_select.kPMC_BandgapBufferDriveHigh (*C enumerator*), 437
drivers_Kinetis_K_MK22F51212._pmc_bandgap_buffer_drive_select.kPMC_BandgapBufferDriveLow (*C enumerator*), 437
drivers_Kinetis_K_MK22F51212._pmc_high_volt_detect_config (*C struct*), 439
drivers_Kinetis_K_MK22F51212._pmc_high_volt_detect_config.enableInt (*C var*), 439
drivers_Kinetis_K_MK22F51212._pmc_high_volt_detect_config.enableReset (*C var*), 439
drivers_Kinetis_K_MK22F51212._pmc_high_volt_detect_config.voltSelect (*C var*), 439
drivers_Kinetis_K_MK22F51212._pmc_high_volt_detect_volt_select (*C enum*), 436
drivers_Kinetis_K_MK22F51212._pmc_high_volt_detect_volt_select.kPMC_HighVoltDetectHighTrip (*C enumerator*), 436
drivers_Kinetis_K_MK22F51212._pmc_high_volt_detect_volt_select.kPMC_HighVoltDetectLowTrip (*C enumerator*), 436

drivers_Kinetis_K_MK22F51212._pmc_low_volt_detect_config (*C struct*), 438
drivers_Kinetis_K_MK22F51212._pmc_low_volt_detect_config.enableInt (*C var*), 438
drivers_Kinetis_K_MK22F51212._pmc_low_volt_detect_config.enableReset (*C var*), 438
drivers_Kinetis_K_MK22F51212._pmc_low_volt_detect_config.voltSelect (*C var*), 438
drivers_Kinetis_K_MK22F51212._pmc_low_volt_detect_volt_select (*C enum*), 436
drivers_Kinetis_K_MK22F51212._pmc_low_volt_detect_volt_select.kPMC_LowVoltDetectHighTrip (*C enumerator*), 436
drivers_Kinetis_K_MK22F51212._pmc_low_volt_detect_volt_select.kPMC_LowVoltDetectLowTrip (*C enumerator*), 436
drivers_Kinetis_K_MK22F51212._pmc_low_volt_warning_config (*C struct*), 438
drivers_Kinetis_K_MK22F51212._pmc_low_volt_warning_config.enableInt (*C var*), 439
drivers_Kinetis_K_MK22F51212._pmc_low_volt_warning_config.voltSelect (*C var*), 439
drivers_Kinetis_K_MK22F51212._pmc_low_volt_warning_volt_select (*C enum*), 436
drivers_Kinetis_K_MK22F51212._pmc_low_volt_warning_volt_select.kPMC_LowVoltWarningHighTrip (*C enumerator*), 436
drivers_Kinetis_K_MK22F51212._pmc_low_volt_warning_volt_select.kPMC_LowVoltWarningLowTrip (*C enumerator*), 436
drivers_Kinetis_K_MK22F51212._pmc_low_volt_warning_volt_select.kPMC_LowVoltWarningMid1Trip (*C enumerator*), 436
drivers_Kinetis_K_MK22F51212._pmc_low_volt_warning_volt_select.kPMC_LowVoltWarningMid2Trip (*C enumerator*), 436
drivers_Kinetis_K_MK22F51212._pmc_param (*C struct*), 438
drivers_Kinetis_K_MK22F51212._pmc_param.hvdEnable (*C var*), 438
drivers_Kinetis_K_MK22F51212._pmc_param.vlpoEnable (*C var*), 438
drivers_Kinetis_K_MK22F51212._pmc_version_id (*C struct*), 438
drivers_Kinetis_K_MK22F51212._pmc_version_id.feature (*C var*), 438
drivers_Kinetis_K_MK22F51212._pmc_version_id.major (*C var*), 438
drivers_Kinetis_K_MK22F51212._pmc_version_id.minor (*C var*), 438
drivers_Kinetis_K_MK22F51212._pmc_vlp_freq_option (*C enum*), 437
drivers_Kinetis_K_MK22F51212._pmc_vlp_freq_option.kPMC_FreqRestrict (*C enumerator*), 437
drivers_Kinetis_K_MK22F51212._pmc_vlp_freq_option.kPMC_FreqUnrestrict (*C enumerator*), 437
drivers_Kinetis_K_MK22F51212._port_digital_filter_clock_source (*C enum*), 446
drivers_Kinetis_K_MK22F51212._port_digital_filter_clock_source.kPORT_BusClock (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_digital_filter_clock_source.kPORT_LpoClock (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_digital_filter_config (*C struct*), 447
drivers_Kinetis_K_MK22F51212._port_digital_filter_config.clockSource (*C var*), 447
drivers_Kinetis_K_MK22F51212._port_digital_filter_config.digitalFilterWidth (*C var*), 447
drivers_Kinetis_K_MK22F51212._port_drive_strength (*C enum*), 444
drivers_Kinetis_K_MK22F51212._port_drive_strength.kPORT_HighDriveStrength (*C enumerator*), 444
drivers_Kinetis_K_MK22F51212._port_drive_strength.kPORT_LowDriveStrength (*C enumerator*), 444
drivers_Kinetis_K_MK22F51212._port_interrupt (*C enum*), 445
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_ActiveHighTriggerOutputEnable (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_ActiveLowTriggerOutputEnable (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_DMAEitherEdge (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_DMAFallingEdge (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_DMARisingEdge (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_FlagEitherEdge (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_FlagFallingEdge (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_FlagRisingEdge (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_InterruptEitherEdge (*C enumerator*), 446

drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_InterruptFallingEdge (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_InterruptLogicOne (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_InterruptLogicZero (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_InterruptOrDMADisabled (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_interrupt.kPORT_InterruptRisingEdge (*C enumerator*), 446
drivers_Kinetis_K_MK22F51212._port_lock_register (*C enum*), 444
drivers_Kinetis_K_MK22F51212._port_lock_register.kPORT_LockRegister (*C enumerator*), 444
drivers_Kinetis_K_MK22F51212._port_lock_register.kPORT_UnlockRegister (*C enumerator*), 444
drivers_Kinetis_K_MK22F51212._port_mux (*C enum*), 444
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt0 (*C enumerator*), 444
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt1 (*C enumerator*), 444
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt2 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt3 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt4 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt5 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt6 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt7 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt8 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt9 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt10 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt11 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt12 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt13 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt14 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAlt15 (*C enumerator*), 445
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_MuxAsGpio (*C enumerator*), 444
drivers_Kinetis_K_MK22F51212._port_mux.kPORT_PinDisabledOrAnalog (*C enumerator*), 444
drivers_Kinetis_K_MK22F51212._port_open_drain_enable (*C enum*), 443
drivers_Kinetis_K_MK22F51212._port_open_drain_enable.kPORT_OpenDrainDisable (*C enumerator*), 443
drivers_Kinetis_K_MK22F51212._port_open_drain_enable.kPORT_OpenDrainEnable (*C enumerator*), 444
drivers_Kinetis_K_MK22F51212._port_passive_filter_enable (*C enum*), 444
drivers_Kinetis_K_MK22F51212._port_passive_filter_enable.kPORT_PassiveFilterDisable (*C enumerator*), 444
drivers_Kinetis_K_MK22F51212._port_passive_filter_enable.kPORT_PassiveFilterEnable (*C enumerator*), 444
drivers_Kinetis_K_MK22F51212._port_pin_config (*C struct*), 447
drivers_Kinetis_K_MK22F51212._port_pin_config.driveStrength (*C var*), 448
drivers_Kinetis_K_MK22F51212._port_pin_config.lockRegister (*C var*), 448
drivers_Kinetis_K_MK22F51212._port_pin_config.openDrainEnable (*C var*), 447
drivers_Kinetis_K_MK22F51212._port_pin_config.passiveFilterEnable (*C var*), 447
drivers_Kinetis_K_MK22F51212._port_pin_config.pullSelect (*C var*), 447
drivers_Kinetis_K_MK22F51212._port_pin_config.slewRate (*C var*), 447
drivers_Kinetis_K_MK22F51212._port_pull (*C enum*), 443
drivers_Kinetis_K_MK22F51212._port_pull.kPORT_PullDisable (*C enumerator*), 443
drivers_Kinetis_K_MK22F51212._port_pull.kPORT_PullDown (*C enumerator*), 443
drivers_Kinetis_K_MK22F51212._port_pull.kPORT_PullUp (*C enumerator*), 443
drivers_Kinetis_K_MK22F51212._port_slew_rate (*C enum*), 443
drivers_Kinetis_K_MK22F51212._port_slew_rate.kPORT_FastSlewRate (*C enumerator*), 443
drivers_Kinetis_K_MK22F51212._port_slew_rate.kPORT_SlowSlewRate (*C enumerator*), 443
drivers_Kinetis_K_MK22F51212._rcm_boot_rom_config (*C enum*), 452
drivers_Kinetis_K_MK22F51212._rcm_boot_rom_config.kRCM_BootFlash (*C enumerator*), 452
drivers_Kinetis_K_MK22F51212._rcm_boot_rom_config.kRCM_BootRomBoth (*C enumerator*), 452

drivers_Kinetis_K_MK22F51212._rcm_boot_rom_config.kRCM_BootRomCfg0 (*C enumerator*), 452
drivers_Kinetis_K_MK22F51212._rcm_boot_rom_config.kRCM_BootRomFopt (*C enumerator*), 452
drivers_Kinetis_K_MK22F51212._rcm_interrupt_enable (*C enum*), 452
drivers_Kinetis_K_MK22F51212._rcm_interrupt_enable.kRCM_IntAll (*C enumerator*), 453
drivers_Kinetis_K_MK22F51212._rcm_interrupt_enable.kRCM_IntCore1 (*C enumerator*), 453
drivers_Kinetis_K_MK22F51212._rcm_interrupt_enable.kRCM_IntCoreLockup (*C enumerator*), 453
drivers_Kinetis_K_MK22F51212._rcm_interrupt_enable.kRCM_IntExternalPin (*C enumerator*), 453
drivers_Kinetis_K_MK22F51212._rcm_interrupt_enable.kRCM_IntGlobal (*C enumerator*), 453
drivers_Kinetis_K_MK22F51212._rcm_interrupt_enable.kRCM_IntLossOfClk (*C enumerator*), 452
drivers_Kinetis_K_MK22F51212._rcm_interrupt_enable.kRCM_IntLossOfLock (*C enumerator*), 453
drivers_Kinetis_K_MK22F51212._rcm_interrupt_enable.kRCM_IntNone (*C enumerator*), 452
drivers_Kinetis_K_MK22F51212._rcm_interrupt_enable.kRCM_IntSoftware (*C enumerator*), 453
drivers_Kinetis_K_MK22F51212._rcm_interrupt_enable.kRCM_IntStopModeAckErr (*C enumerator*), 453
drivers_Kinetis_K_MK22F51212._rcm_interrupt_enable.kRCM_IntWatchDog (*C enumerator*), 453
drivers_Kinetis_K_MK22F51212._rcm_reset_delay (*C enum*), 452
drivers_Kinetis_K_MK22F51212._rcm_reset_delay.kRCM_ResetDelay8Lpo (*C enumerator*), 452
drivers_Kinetis_K_MK22F51212._rcm_reset_delay.kRCM_ResetDelay32Lpo (*C enumerator*), 452
drivers_Kinetis_K_MK22F51212._rcm_reset_delay.kRCM_ResetDelay128Lpo (*C enumerator*), 452
drivers_Kinetis_K_MK22F51212._rcm_reset_delay.kRCM_ResetDelay512Lpo (*C enumerator*), 452
drivers_Kinetis_K_MK22F51212._rcm_reset_pin_filter_config (*C struct*), 454
drivers_Kinetis_K_MK22F51212._rcm_reset_pin_filter_config.busClockFilterCount (*C var*), 454
drivers_Kinetis_K_MK22F51212._rcm_reset_pin_filter_config.enableFilterInStop (*C var*), 454
drivers_Kinetis_K_MK22F51212._rcm_reset_pin_filter_config.filterInRunWait (*C var*), 454
drivers_Kinetis_K_MK22F51212._rcm_reset_source (*C enum*), 450
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourceAll (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourceEzpt (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourceJtag (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourceLoc (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourceLockup (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourceLol (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourceLvd (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourceMdmap (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourcePin (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourcePor (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourceSackerr (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourceSw (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourceWakeup (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_reset_source.kRCM_SourceWdog (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_run_wait_filter_mode (*C enum*), 451
drivers_Kinetis_K_MK22F51212._rcm_run_wait_filter_mode.kRCM_FilterBusClock (*C enumerator*), 452
drivers_Kinetis_K_MK22F51212._rcm_run_wait_filter_mode.kRCM_FilterDisable (*C enumerator*), 451
drivers_Kinetis_K_MK22F51212._rcm_run_wait_filter_mode.kRCM_FilterLpoClock (*C enumerator*), 452
drivers_Kinetis_K_MK22F51212._rcm_version_id (*C struct*), 454
drivers_Kinetis_K_MK22F51212._rcm_version_id.feature (*C var*), 454
drivers_Kinetis_K_MK22F51212._rcm_version_id.major (*C var*), 454
drivers_Kinetis_K_MK22F51212._rcm_version_id.minor (*C var*), 454
drivers_Kinetis_K_MK22F51212._rnga_mode (*C enum*), 454
drivers_Kinetis_K_MK22F51212._rnga_mode.kRNGA_ModeNormal (*C enumerator*), 454
drivers_Kinetis_K_MK22F51212._rnga_mode.kRNGA_ModeSleep (*C enumerator*), 454

drivers_Kinetis_K_MK22F51212._rtc_config (*C struct*), 464
drivers_Kinetis_K_MK22F51212._rtc_config.clockOutput (*C var*), 464
drivers_Kinetis_K_MK22F51212._rtc_config.compensationInterval (*C var*), 464
drivers_Kinetis_K_MK22F51212._rtc_config.compensationTime (*C var*), 464
drivers_Kinetis_K_MK22F51212._rtc_config.supervisorAccess (*C var*), 464
drivers_Kinetis_K_MK22F51212._rtc_config.updateMode (*C var*), 464
drivers_Kinetis_K_MK22F51212._rtc_config.wakeupSelect (*C var*), 464
drivers_Kinetis_K_MK22F51212._rtc_datetime (*C struct*), 463
drivers_Kinetis_K_MK22F51212._rtc_datetime.day (*C var*), 463
drivers_Kinetis_K_MK22F51212._rtc_datetime.hour (*C var*), 463
drivers_Kinetis_K_MK22F51212._rtc_datetime.minute (*C var*), 463
drivers_Kinetis_K_MK22F51212._rtc_datetime.month (*C var*), 463
drivers_Kinetis_K_MK22F51212._rtc_datetime.second (*C var*), 463
drivers_Kinetis_K_MK22F51212._rtc_datetime.year (*C var*), 463
drivers_Kinetis_K_MK22F51212._rtc_interrupt_enable (*C enum*), 459
drivers_Kinetis_K_MK22F51212._rtc_interrupt_enable.kRTC_AlarmInterruptEnable (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_interrupt_enable.kRTC_FlashSecurityInterruptEnable (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_interrupt_enable.kRTC_LossOfClockInterruptEnable (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_interrupt_enable.kRTC_MonotonicOverflowInterruptEnable (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_interrupt_enable.kRTC_SecondsInterruptEnable (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_interrupt_enable.kRTC_SecurityModuleInterruptEnable (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_interrupt_enable.kRTC_TamperPinInterruptEnable (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_interrupt_enable.kRTC_TestModeInterruptEnable (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_interrupt_enable.kRTC_TimeInvalidInterruptEnable (*C enumerator*), 459
drivers_Kinetis_K_MK22F51212._rtc_interrupt_enable.kRTC_TimeOverflowInterruptEnable (*C enumerator*), 459
drivers_Kinetis_K_MK22F51212._rtc_osc_cap_load (*C enum*), 461
drivers_Kinetis_K_MK22F51212._rtc_osc_cap_load.kRTC_Capacitor_2p (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._rtc_osc_cap_load.kRTC_Capacitor_4p (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._rtc_osc_cap_load.kRTC_Capacitor_8p (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._rtc_osc_cap_load.kRTC_Capacitor_16p (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._rtc_pin_config (*C struct*), 463
drivers_Kinetis_K_MK22F51212._rtc_pin_config.filterEnable (*C var*), 464
drivers_Kinetis_K_MK22F51212._rtc_pin_config.inputLogic (*C var*), 464
drivers_Kinetis_K_MK22F51212._rtc_pin_config.pinActiveLow (*C var*), 464
drivers_Kinetis_K_MK22F51212._rtc_pin_config.pullEnable (*C var*), 464
drivers_Kinetis_K_MK22F51212._rtc_pin_config.pullSelectNegate (*C var*), 464
drivers_Kinetis_K_MK22F51212._rtc_status_flags (*C enum*), 460
drivers_Kinetis_K_MK22F51212._rtc_status_flags.kRTC_AlarmFlag (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_status_flags.kRTC_FlashSecurityFlag (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_status_flags.kRTC_LossOfClockTamperFlag (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._rtc_status_flags.kRTC_MonotonicOverflowFlag (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_status_flags.kRTC_SecurityTamperFlag (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_status_flags.kRTC_TamperInterruptDetectFlag (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_status_flags.kRTC_TamperPinFlag (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_status_flags.kRTC_TestModeFlag (*C enumerator*), 460

drivers_Kinetis_K_MK22F51212._rtc_status_flags.kRTC_TimeInvalidFlag (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_status_flags.kRTC_TimeOverflowFlag (*C enumerator*), 460
drivers_Kinetis_K_MK22F51212._rtc_timer_seconds_interrupt_frequency (*C enum*), 461
drivers_Kinetis_K_MK22F51212._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency1Hz (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency2Hz (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency4Hz (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency8Hz (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency16Hz (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency32Hz (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency64Hz (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._rtc_timer_seconds_interrupt_frequency.kRTC_TimerSecondsFrequency128Hz (*C enumerator*), 461
drivers_Kinetis_K_MK22F51212._sai_bclk_source (*C enum*), 482
drivers_Kinetis_K_MK22F51212._sai_bclk_source.kSAI_BclkSourceBusclk (*C enumerator*), 483
drivers_Kinetis_K_MK22F51212._sai_bclk_source.kSAI_BclkSourceMclkDiv (*C enumerator*), 483
drivers_Kinetis_K_MK22F51212._sai_bclk_source.kSAI_BclkSourceMclkOption1 (*C enumerator*), 483
drivers_Kinetis_K_MK22F51212._sai_bclk_source.kSAI_BclkSourceMclkOption2 (*C enumerator*), 483
drivers_Kinetis_K_MK22F51212._sai_bclk_source.kSAI_BclkSourceMclkOption3 (*C enumerator*), 483
drivers_Kinetis_K_MK22F51212._sai_bclk_source.kSAI_BclkSourceOtherSai0 (*C enumerator*), 483
drivers_Kinetis_K_MK22F51212._sai_bclk_source.kSAI_BclkSourceOtherSai1 (*C enumerator*), 483
drivers_Kinetis_K_MK22F51212._sai_bit_clock (*C struct*), 491
drivers_Kinetis_K_MK22F51212._sai_bit_clock.bclkInputDelay (*C var*), 491
drivers_Kinetis_K_MK22F51212._sai_bit_clock.bclkPolarity (*C var*), 491
drivers_Kinetis_K_MK22F51212._sai_bit_clock.bclkSource (*C var*), 491
drivers_Kinetis_K_MK22F51212._sai_clock_polarity (*C enum*), 482
drivers_Kinetis_K_MK22F51212._sai_clock_polarity.kSAI_PolarityActiveHigh (*C enumerator*), 482
drivers_Kinetis_K_MK22F51212._sai_clock_polarity.kSAI_PolarityActiveLow (*C enumerator*), 482
drivers_Kinetis_K_MK22F51212._sai_clock_polarity.kSAI_SampleOnFallingEdge (*C enumerator*), 482
drivers_Kinetis_K_MK22F51212._sai_clock_polarity.kSAI_SampleOnRisingEdge (*C enumerator*), 482
drivers_Kinetis_K_MK22F51212._sai_config (*C struct*), 489
drivers_Kinetis_K_MK22F51212._sai_config.bclkSource (*C var*), 489
drivers_Kinetis_K_MK22F51212._sai_config.masterSlave (*C var*), 489
drivers_Kinetis_K_MK22F51212._sai_config.mclkOutputEnable (*C var*), 489
drivers_Kinetis_K_MK22F51212._sai_config.mclkSource (*C var*), 489
drivers_Kinetis_K_MK22F51212._sai_config.protocol (*C var*), 489
drivers_Kinetis_K_MK22F51212._sai_config.syncMode (*C var*), 489
drivers_Kinetis_K_MK22F51212._sai_data_order (*C enum*), 481
drivers_Kinetis_K_MK22F51212._sai_data_order.kSAI_DataLSB (*C enumerator*), 481
drivers_Kinetis_K_MK22F51212._sai_data_order.kSAI_DataMSB (*C enumerator*), 482
drivers_Kinetis_K_MK22F51212._sai_data_pin_state (*C enum*), 486
drivers_Kinetis_K_MK22F51212._sai_data_pin_state.kSAI_DataPinStateOutputZero (*C enumerator*), 486
drivers_Kinetis_K_MK22F51212._sai_data_pin_state.kSAI_DataPinStateTriState (*C enumerator*), 486
drivers_Kinetis_K_MK22F51212._sai_edma_interleave (*C enum*), 500
drivers_Kinetis_K_MK22F51212._sai_edma_interleave.kSAI_EDMAInterleavePerChannelBlock (*C*

enumerator), 500
drivers_Kinetis_K_MK22F51212._sai_edma_interleave.kSAI_EDMAInterleavePerChannelSample (*C enumerator*), 500
drivers_Kinetis_K_MK22F51212._sai_fifo (*C struct*), 491
drivers_Kinetis_K_MK22F51212._sai_fifo_combine (*C enum*), 486
drivers_Kinetis_K_MK22F51212._sai_fifo_combine.kSAI_FifoCombineDisabled (*C enumerator*), 486
drivers_Kinetis_K_MK22F51212._sai_fifo_combine.kSAI_FifoCombineModeEnabledOnRead (*C enumerator*), 486
drivers_Kinetis_K_MK22F51212._sai_fifo_combine.kSAI_FifoCombineModeEnabledOnReadWrite (*C enumerator*), 486
drivers_Kinetis_K_MK22F51212._sai_fifo_combine.kSAI_FifoCombineModeEnabledOnWrite (*C enumerator*), 486
drivers_Kinetis_K_MK22F51212._sai_fifo_combine.kSAI_RXFifoCombineModeEnabledOnRead (*C enumerator*), 486
drivers_Kinetis_K_MK22F51212._sai_fifo_combine.kSAI_RxFifoCombineModeEnabledOnWrite (*C enumerator*), 486
drivers_Kinetis_K_MK22F51212._sai_fifo_packing (*C enum*), 484
drivers_Kinetis_K_MK22F51212._sai_fifo_packing.kSAI_FifoPacking8bit (*C enumerator*), 484
drivers_Kinetis_K_MK22F51212._sai_fifo_packing.kSAI_FifoPacking16bit (*C enumerator*), 484
drivers_Kinetis_K_MK22F51212._sai_fifo_packing.kSAI_FifoPackingDisabled (*C enumerator*), 484
drivers_Kinetis_K_MK22F51212._sai_fifo.fifoCombine (*C var*), 491
drivers_Kinetis_K_MK22F51212._sai_fifo.fifoContinueOneError (*C var*), 491
drivers_Kinetis_K_MK22F51212._sai_fifo.fifoPacking (*C var*), 491
drivers_Kinetis_K_MK22F51212._sai_fifo.fifoWatermark (*C var*), 491
drivers_Kinetis_K_MK22F51212._sai_frame_sync (*C struct*), 491
drivers_Kinetis_K_MK22F51212._sai_frame_sync_len (*C enum*), 487
drivers_Kinetis_K_MK22F51212._sai_frame_sync_len.kSAI_FrameSyncLenOneBitClk (*C enumerator*), 487
drivers_Kinetis_K_MK22F51212._sai_frame_sync_len.kSAI_FrameSyncLenPerWordWidth (*C enumerator*), 487
drivers_Kinetis_K_MK22F51212._sai_frame_sync.frameSyncEarly (*C var*), 491
drivers_Kinetis_K_MK22F51212._sai_frame_sync.frameSyncGenerateOnDemand (*C var*), 492
drivers_Kinetis_K_MK22F51212._sai_frame_sync.frameSyncPolarity (*C var*), 492
drivers_Kinetis_K_MK22F51212._sai_frame_sync.frameSyncWidth (*C var*), 491
drivers_Kinetis_K_MK22F51212._sai_handle (*C struct*), 493
drivers_Kinetis_K_MK22F51212._sai_handle.base (*C var*), 493
drivers_Kinetis_K_MK22F51212._sai_handle.bitWidth (*C var*), 494
drivers_Kinetis_K_MK22F51212._sai_handle.callback (*C var*), 494
drivers_Kinetis_K_MK22F51212._sai_handle.channel (*C var*), 494
drivers_Kinetis_K_MK22F51212._sai_handle.channelMask (*C var*), 494
drivers_Kinetis_K_MK22F51212._sai_handle.channelNums (*C var*), 494
drivers_Kinetis_K_MK22F51212._sai_handle.endChannel (*C var*), 494
drivers_Kinetis_K_MK22F51212._sai_handle.queueDriver (*C var*), 494
drivers_Kinetis_K_MK22F51212._sai_handle.queueUser (*C var*), 494
drivers_Kinetis_K_MK22F51212._sai_handle.saiQueue (*C var*), 494
drivers_Kinetis_K_MK22F51212._sai_handle.state (*C var*), 493
drivers_Kinetis_K_MK22F51212._sai_handle.transferSize (*C var*), 494
drivers_Kinetis_K_MK22F51212._sai_handle.userData (*C var*), 494
drivers_Kinetis_K_MK22F51212._sai_handle.watermark (*C var*), 494
drivers_Kinetis_K_MK22F51212._sai_master_clock (*C struct*), 490
drivers_Kinetis_K_MK22F51212._sai_master_clock.mclkHz (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_master_clock.mclkOutputEnable (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_master_clock.mclkSource (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_master_clock.mclkSourceClkHz (*C var*), 491
drivers_Kinetis_K_MK22F51212._sai_master_slave (*C enum*), 481
drivers_Kinetis_K_MK22F51212._sai_master_slave.kSAI_Bclk_Master_FrameSync_Slave (*C enumerator*), 481

drivers_Kinetis_K_MK22F51212._sai_master_slave.kSAI_Bclk_Slave_FrameSync_Master (*C enumerator*), 481
drivers_Kinetis_K_MK22F51212._sai_master_slave.kSAI_Master (*C enumerator*), 481
drivers_Kinetis_K_MK22F51212._sai_master_slave.kSAI_Slave (*C enumerator*), 481
drivers_Kinetis_K_MK22F51212._sai_mclk_source (*C enum*), 482
drivers_Kinetis_K_MK22F51212._sai_mclk_source.kSAI_MclkSourceSelect1 (*C enumerator*), 482
drivers_Kinetis_K_MK22F51212._sai_mclk_source.kSAI_MclkSourceSelect2 (*C enumerator*), 482
drivers_Kinetis_K_MK22F51212._sai_mclk_source.kSAI_MclkSourceSelect3 (*C enumerator*), 482
drivers_Kinetis_K_MK22F51212._sai_mclk_source.kSAI_MclkSourceSysclk (*C enumerator*), 482
drivers_Kinetis_K_MK22F51212._sai_mono_stereo (*C enum*), 481
drivers_Kinetis_K_MK22F51212._sai_mono_stereo.kSAI_MonoLeft (*C enumerator*), 481
drivers_Kinetis_K_MK22F51212._sai_mono_stereo.kSAI_MonoRight (*C enumerator*), 481
drivers_Kinetis_K_MK22F51212._sai_mono_stereo.kSAI_Stereo (*C enumerator*), 481
drivers_Kinetis_K_MK22F51212._sai_protocol (*C enum*), 480
drivers_Kinetis_K_MK22F51212._sai_protocol.kSAI_BusI2S (*C enumerator*), 481
drivers_Kinetis_K_MK22F51212._sai_protocol.kSAI_BusLeftJustified (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212._sai_protocol.kSAI_BusPCMA (*C enumerator*), 481
drivers_Kinetis_K_MK22F51212._sai_protocol.kSAI_BusPCMB (*C enumerator*), 481
drivers_Kinetis_K_MK22F51212._sai_protocol.kSAI_BusRightJustified (*C enumerator*), 481
drivers_Kinetis_K_MK22F51212._sai_reset_type (*C enum*), 484
drivers_Kinetis_K_MK22F51212._sai_reset_type.kSAI_ResetAll (*C enumerator*), 484
drivers_Kinetis_K_MK22F51212._sai_reset_type.kSAI_ResetTypeFIFO (*C enumerator*), 484
drivers_Kinetis_K_MK22F51212._sai_reset_type.kSAI_ResetTypeSoftware (*C enumerator*), 484
drivers_Kinetis_K_MK22F51212._sai_sample_rate (*C enum*), 485
drivers_Kinetis_K_MK22F51212._sai_sample_rate.kSAI_SampleRate8KHz (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_sample_rate.kSAI_SampleRate12KHz (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_sample_rate.kSAI_SampleRate16KHz (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_sample_rate.kSAI_SampleRate24KHz (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_sample_rate.kSAI_SampleRate32KHz (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_sample_rate.kSAI_SampleRate48KHz (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_sample_rate.kSAI_SampleRate96KHz (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_sample_rate.kSAI_SampleRate192KHz (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_sample_rate.kSAI_SampleRate384KHz (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_sample_rate.kSAI_SampleRate11025Hz (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_sample_rate.kSAI_SampleRate22050Hz (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_sample_rate.kSAI_SampleRate44100Hz (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_serial_data (*C struct*), 492
drivers_Kinetis_K_MK22F51212._sai_serial_data.dataFirstBitShifted (*C var*), 492
drivers_Kinetis_K_MK22F51212._sai_serial_data.dataMaskedWord (*C var*), 492
drivers_Kinetis_K_MK22F51212._sai_serial_data.dataMode (*C var*), 492
drivers_Kinetis_K_MK22F51212._sai_serial_data.dataOrder (*C var*), 492
drivers_Kinetis_K_MK22F51212._sai_serial_data.dataWord0Length (*C var*), 492
drivers_Kinetis_K_MK22F51212._sai_serial_data.dataWordLength (*C var*), 492
drivers_Kinetis_K_MK22F51212._sai_serial_data.dataWordNLength (*C var*), 492
drivers_Kinetis_K_MK22F51212._sai_serial_data.dataWordNum (*C var*), 492
drivers_Kinetis_K_MK22F51212._sai_sync_mode (*C enum*), 482
drivers_Kinetis_K_MK22F51212._sai_sync_mode.kSAI_ModeAsync (*C enumerator*), 482
drivers_Kinetis_K_MK22F51212._sai_sync_mode.kSAI_ModeSync (*C enumerator*), 482
drivers_Kinetis_K_MK22F51212._sai_transceiver (*C struct*), 492
drivers_Kinetis_K_MK22F51212._sai_transceiver_type (*C enum*), 486
drivers_Kinetis_K_MK22F51212._sai_transceiver_type.kSAI_Receiver (*C enumerator*), 487
drivers_Kinetis_K_MK22F51212._sai_transceiver_type.kSAI_Transmitter (*C enumerator*), 486
drivers_Kinetis_K_MK22F51212._sai_transceiver.bitClock (*C var*), 493
drivers_Kinetis_K_MK22F51212._sai_transceiver.channelMask (*C var*), 493
drivers_Kinetis_K_MK22F51212._sai_transceiver.channelNums (*C var*), 493
drivers_Kinetis_K_MK22F51212._sai_transceiver.endChannel (*C var*), 493
drivers_Kinetis_K_MK22F51212._sai_transceiver.fifo (*C var*), 493
drivers_Kinetis_K_MK22F51212._sai_transceiver.frameSync (*C var*), 492

drivers_Kinetis_K_MK22F51212._sai_transceiver.masterSlave (*C var*), 493
drivers_Kinetis_K_MK22F51212._sai_transceiver.serialData (*C var*), 492
drivers_Kinetis_K_MK22F51212._sai_transceiver.startChannel (*C var*), 493
drivers_Kinetis_K_MK22F51212._sai_transceiver.syncMode (*C var*), 493
drivers_Kinetis_K_MK22F51212._sai_transfer (*C struct*), 493
drivers_Kinetis_K_MK22F51212._sai_transfer_format (*C struct*), 489
drivers_Kinetis_K_MK22F51212._sai_transfer_format.bitWidth (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_transfer_format.channel (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_transfer_format.channelMask (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_transfer_format.channelNums (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_transfer_format.endChannel (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_transfer_format.isFrameSyncCompact (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_transfer_format.masterClockHz (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_transfer_format.protocol (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_transfer_format.sampleRate_Hz (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_transfer_format.stereo (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_transfer_format.watermark (*C var*), 490
drivers_Kinetis_K_MK22F51212._sai_transfer.data (*C var*), 493
drivers_Kinetis_K_MK22F51212._sai_transfer.dataSize (*C var*), 493
drivers_Kinetis_K_MK22F51212._sai_word_width (*C enum*), 485
drivers_Kinetis_K_MK22F51212._sai_word_width.kSAI_WordWidth8bits (*C enumerator*), 485
drivers_Kinetis_K_MK22F51212._sai_word_width.kSAI_WordWidth16bits (*C enumerator*), 486
drivers_Kinetis_K_MK22F51212._sai_word_width.kSAI_WordWidth24bits (*C enumerator*), 486
drivers_Kinetis_K_MK22F51212._sai_word_width.kSAI_WordWidth32bits (*C enumerator*), 486
drivers_Kinetis_K_MK22F51212._SDK_AtomicLocalCompareAndSet (*C function*), 367
drivers_Kinetis_K_MK22F51212._SDK_AtomicTestAndSet (*C function*), 367
drivers_Kinetis_K_MK22F51212._sim_clock_config (*C struct*), 176
drivers_Kinetis_K_MK22F51212._sim_flash_mode (*C enum*), 502
drivers_Kinetis_K_MK22F51212._sim_flash_mode.kSIM_FlashDisable (*C enumerator*), 502
drivers_Kinetis_K_MK22F51212._sim_flash_mode.kSIM_FlashDisableInWait (*C enumerator*), 502
drivers_Kinetis_K_MK22F51212._sim_uid (*C struct*), 502
drivers_Kinetis_K_MK22F51212._sim_uid.H (*C var*), 503
drivers_Kinetis_K_MK22F51212._sim_uid.L (*C var*), 503
drivers_Kinetis_K_MK22F51212._sim_uid.M (*C var*), 503
drivers_Kinetis_K_MK22F51212._sim_usb_volt_reg_enable_mode (*C enum*), 501
drivers_Kinetis_K_MK22F51212._sim_usb_volt_reg_enable_mode.kSIM_UsbVoltRegEnable (*C enumerator*), 502
drivers_Kinetis_K_MK22F51212._sim_usb_volt_reg_enable_mode.kSIM_UsbVoltRegEnableInAllModes (*C enumerator*), 502
drivers_Kinetis_K_MK22F51212._sim_usb_volt_reg_enable_mode.kSIM_UsbVoltRegEnableInLowPower (*C enumerator*), 502
drivers_Kinetis_K_MK22F51212._sim_usb_volt_reg_enable_mode.kSIM_UsbVoltRegEnableInStop (*C enumerator*), 502
drivers_Kinetis_K_MK22F51212._smartcard_card_convention (*C enum*), 506
drivers_Kinetis_K_MK22F51212._smartcard_card_convention.kSMARTCARD_DirectConvention (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_card_convention.kSMARTCARD_InverseConvention (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_card_params (*C struct*), 508
drivers_Kinetis_K_MK22F51212._smartcard_card_params.active (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_card_params.atrComplete (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_card_params.atrValid (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_card_params.BGI (*C var*), 508
drivers_Kinetis_K_MK22F51212._smartcard_card_params.BWI (*C var*), 508
drivers_Kinetis_K_MK22F51212._smartcard_card_params.convention (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_card_params.currentD (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_card_params.CWI (*C var*), 508
drivers_Kinetis_K_MK22F51212._smartcard_card_params.Di (*C var*), 508

drivers_Kinetis_K_MK22F51212._smartcard_card_params.faulty (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_card_params.Fi (*C var*), 508
drivers_Kinetis_K_MK22F51212._smartcard_card_params.fMax (*C var*), 508
drivers_Kinetis_K_MK22F51212._smartcard_card_params.GTN (*C var*), 508
drivers_Kinetis_K_MK22F51212._smartcard_card_params.IFSC (*C var*), 508
drivers_Kinetis_K_MK22F51212._smartcard_card_params.modeNegotiable (*C var*), 508
drivers_Kinetis_K_MK22F51212._smartcard_card_params.present (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_card_params.status (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_card_params.t0Indicated (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_card_params.t1Indicated (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_card_params.WI (*C var*), 508
drivers_Kinetis_K_MK22F51212._smartcard_card_voltage_class (*C enum*), 505
drivers_Kinetis_K_MK22F51212._smartcard_card_voltage_class.kSMARTCARD_VoltageClassA5_0V (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_card_voltage_class.kSMARTCARD_VoltageClassB3_3V (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_card_voltage_class.kSMARTCARD_VoltageClassC1_8V (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_card_voltage_class.kSMARTCARD_VoltageClassUnknown (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_context (*C struct*), 511
drivers_Kinetis_K_MK22F51212._smartcard_context.abortTransfer (*C var*), 513
drivers_Kinetis_K_MK22F51212._smartcard_context.base (*C var*), 511
drivers_Kinetis_K_MK22F51212._smartcard_context.cardParams (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.direction (*C var*), 511
drivers_Kinetis_K_MK22F51212._smartcard_context.IFSD (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.interfaceCallback (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.interfaceCallbackParam (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.interfaceConfig (*C var*), 513
drivers_Kinetis_K_MK22F51212._smartcard_context.parity (*C var*), 513
drivers_Kinetis_K_MK22F51212._smartcard_context.parityError (*C var*), 513
drivers_Kinetis_K_MK22F51212._smartcard_context.resetType (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.rxFifoThreshold (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.rxtCrossed (*C var*), 513
drivers_Kinetis_K_MK22F51212._smartcard_context.statusBytes (*C var*), 513
drivers_Kinetis_K_MK22F51212._smartcard_context.timeDelay (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.timersState (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.transferCallback (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.transferCallbackParam (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.transferState (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.tType (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.txFifoEntryCount (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.txtCrossed (*C var*), 513
drivers_Kinetis_K_MK22F51212._smartcard_context.wtxRequested (*C var*), 513
drivers_Kinetis_K_MK22F51212._smartcard_context.xBuff (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.xIsBusy (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_context.xSize (*C var*), 512
drivers_Kinetis_K_MK22F51212._smartcard_control (*C enum*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_ConfigureBaudrate (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_DisableADT (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_DisableAnack (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_DisableBWT (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_DisableCWT (*C enumerator*), 504

drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_DisableGTV (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_DisableReceiverMode (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_DisableTransmitterMode (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_DisableWWT (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_EnableADT (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_EnableAnack (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_EnableBWT (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_EnableCWT (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_EnableGTV (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_EnableInitDetect (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_EnableReceiverMode (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_EnableTransmitterMode (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_EnableWWT (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_ResetBWT (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_ResetCWT (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_ResetWaitTimeMultiplier (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_ResetWWT (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_SetupATRMode (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_SetupT0Mode (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212._smartcard_control.kSMARTCARD_SetupT1Mode (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_direction (*C enum*), 506
drivers_Kinetis_K_MK22F51212._smartcard_direction.kSMARTCARD_Receive (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_direction.kSMARTCARD_Transmit (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_interface_config (*C struct*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.clockModule (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.clockModuleChannel (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.clockModuleSourceClock (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.clockToResetDelay (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.controlPin (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.controlPort (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.dataPin (*C var*), 511
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.dataPinMux (*C var*), 511
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.dataPort (*C var*), 511
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.irqPin (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.irqPort (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.resetPin (*C var*), 510

drivers_Kinetis_K_MK22F51212._smartcard_interface_config.resetPort (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.smartCardClock (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.tsTimerId (*C var*), 511
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.vcc (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.vsel0Pin (*C var*), 511
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.vsel0Port (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.vsel1Pin (*C var*), 511
drivers_Kinetis_K_MK22F51212._smartcard_interface_config.vsel1Port (*C var*), 511
drivers_Kinetis_K_MK22F51212._smartcard_interface_control (*C enum*), 506
drivers_Kinetis_K_MK22F51212._smartcard_interface_control.kSMARTCARD_InterfaceReadStatus (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_interface_control.kSMARTCARD_InterfaceSetClockToResetDelay (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_interface_control.kSMARTCARD_InterfaceSetVcc (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_parity_type (*C enum*), 506
drivers_Kinetis_K_MK22F51212._smartcard_parity_type.kSMARTCARD_EvenParity (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_parity_type.kSMARTCARD_OddParity (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_reset_type (*C enum*), 505
drivers_Kinetis_K_MK22F51212._smartcard_reset_type.kSMARTCARD_ColdReset (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_reset_type.kSMARTCARD_NoColdReset (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_reset_type.kSMARTCARD_NoWarmReset (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_reset_type.kSMARTCARD_WarmReset (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_timers_state (*C struct*), 509
drivers_Kinetis_K_MK22F51212._smartcard_timers_state.adtExpired (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_timers_state.bwtExpired (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_timers_state.cwtExpired (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_timers_state.initCharTimerExpired (*C var*), 510
drivers_Kinetis_K_MK22F51212._smartcard_timers_state.wwtExpired (*C var*), 509
drivers_Kinetis_K_MK22F51212._smartcard_transfer_state (*C enum*), 505
drivers_Kinetis_K_MK22F51212._smartcard_transfer_state.kSMARTCARD_IdleState (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_transfer_state.kSMARTCARD_InvalidTSDetecetedState (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_transfer_state.kSMARTCARD_ReceivingState (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_transfer_state.kSMARTCARD_TransmittingState (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_transfer_state.kSMARTCARD_WaitingForTSState (*C enumerator*), 505
drivers_Kinetis_K_MK22F51212._smartcard_transport_type (*C enum*), 506
drivers_Kinetis_K_MK22F51212._smartcard_transport_type.kSMARTCARD_T0Transport (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_transport_type.kSMARTCARD_T1Transport (*C enumerator*), 506
drivers_Kinetis_K_MK22F51212._smartcard_xfer (*C struct*), 511
drivers_Kinetis_K_MK22F51212._smartcard_xfer.buff (*C var*), 511
drivers_Kinetis_K_MK22F51212._smartcard_xfer.direction (*C var*), 511
drivers_Kinetis_K_MK22F51212._smartcard_xfer.size (*C var*), 511
drivers_Kinetis_K_MK22F51212._smc_param (*C struct*), 524
drivers_Kinetis_K_MK22F51212._smc_param.hsrunEnable (*C var*), 524
drivers_Kinetis_K_MK22F51212._smc_param.lls2Enable (*C var*), 524

drivers_Kinetis_K_MK22F51212._smc_param.llsEnable (*C var*), 524
drivers_Kinetis_K_MK22F51212._smc_param.vlls0Enable (*C var*), 524
drivers_Kinetis_K_MK22F51212._smc_partial_stop_mode (*C enum*), 522
drivers_Kinetis_K_MK22F51212._smc_partial_stop_mode.kSMC_PartialStop (*C enumerator*), 522
drivers_Kinetis_K_MK22F51212._smc_partial_stop_mode.kSMC_PartialStop1 (*C enumerator*), 523
drivers_Kinetis_K_MK22F51212._smc_partial_stop_mode.kSMC_PartialStop2 (*C enumerator*), 523
drivers_Kinetis_K_MK22F51212._smc_power_mode_lls_config (*C struct*), 524
drivers_Kinetis_K_MK22F51212._smc_power_mode_lls_config.enableLpoClock (*C var*), 524
drivers_Kinetis_K_MK22F51212._smc_power_mode_lls_config.subMode (*C var*), 524
drivers_Kinetis_K_MK22F51212._smc_power_mode_protection (*C enum*), 520
drivers_Kinetis_K_MK22F51212._smc_power_mode_protection.kSMC_AllowPowerModeAll (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_power_mode_protection.kSMC_AllowPowerModeHsrun (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_power_mode_protection.kSMC_AllowPowerModeLls (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_power_mode_protection.kSMC_AllowPowerModeVlls (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_power_mode_protection.kSMC_AllowPowerModeVlp (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_power_mode_vlls_config (*C struct*), 524
drivers_Kinetis_K_MK22F51212._smc_power_mode_vlls_config.enableLpoClock (*C var*), 525
drivers_Kinetis_K_MK22F51212._smc_power_mode_vlls_config.enablePorDetectInVlls0 (*C var*), 524
drivers_Kinetis_K_MK22F51212._smc_power_mode_vlls_config.enableRam2InVlls2 (*C var*), 525
drivers_Kinetis_K_MK22F51212._smc_power_mode_vlls_config.subMode (*C var*), 524
drivers_Kinetis_K_MK22F51212._smc_power_state (*C enum*), 521
drivers_Kinetis_K_MK22F51212._smc_power_state.kSMC_PowerStateHsrun (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_power_state.kSMC_PowerStateLls (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_power_state.kSMC_PowerStateRun (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_power_state.kSMC_PowerStateStop (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_power_state.kSMC_PowerStateVlls (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_power_state.kSMC_PowerStateVlpr (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_power_state.kSMC_PowerStateVlps (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_power_state.kSMC_PowerStateVlpw (*C enumerator*), 521
drivers_Kinetis_K_MK22F51212._smc_run_mode (*C enum*), 521
drivers_Kinetis_K_MK22F51212._smc_run_mode.kSMC_Hsrun (*C enumerator*), 522
drivers_Kinetis_K_MK22F51212._smc_run_mode.kSMC_RunNormal (*C enumerator*), 522
drivers_Kinetis_K_MK22F51212._smc_run_mode.kSMC_RunVlpr (*C enumerator*), 522
drivers_Kinetis_K_MK22F51212._smc_stop_mode (*C enum*), 522
drivers_Kinetis_K_MK22F51212._smc_stop_mode.kSMC_StopLls (*C enumerator*), 522
drivers_Kinetis_K_MK22F51212._smc_stop_mode.kSMC_StopNormal (*C enumerator*), 522
drivers_Kinetis_K_MK22F51212._smc_stop_mode.kSMC_StopVlls (*C enumerator*), 522
drivers_Kinetis_K_MK22F51212._smc_stop_mode.kSMC_StopVlps (*C enumerator*), 522
drivers_Kinetis_K_MK22F51212._smc_stop_submode (*C enum*), 522
drivers_Kinetis_K_MK22F51212._smc_stop_submode.kSMC_StopSub0 (*C enumerator*), 522
drivers_Kinetis_K_MK22F51212._smc_stop_submode.kSMC_StopSub1 (*C enumerator*), 522
drivers_Kinetis_K_MK22F51212._smc_stop_submode.kSMC_StopSub2 (*C enumerator*), 522
drivers_Kinetis_K_MK22F51212._smc_stop_submode.kSMC_StopSub3 (*C enumerator*), 522
drivers_Kinetis_K_MK22F51212._smc_version_id (*C struct*), 523
drivers_Kinetis_K_MK22F51212._smc_version_id.feature (*C var*), 524
drivers_Kinetis_K_MK22F51212._smc_version_id.major (*C var*), 524
drivers_Kinetis_K_MK22F51212._smc_version_id.minor (*C var*), 524
drivers_Kinetis_K_MK22F51212._status_groups (*C enum*), 355
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_AON_COMMON (*C enumerator*), 363

drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_AON_POWER (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_ApplicationRangeStart (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_ASRC (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_AUDMIX (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_BMA (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_BUTTON (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_CAAM (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_CLIF (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_CODEC (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_COMMON_TASK (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_CSI (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_CSS_PKC (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_DCP (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_DebugConsole (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_DMA (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_DMAMGR (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_DMIC (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_DSPI (*C enumerator*), 355
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_ECSPI (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_EDMA (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_ELA_CSEC (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_ELE (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_ELEMU (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_ELS_PKC (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_ENDAT3 (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_ENET (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_ENET_QOS (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_ESAI (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_EVENT_CTRL (*C enumerator*), 364
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_EXTERN_EEPROM (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLASH (*C enumerator*), 355
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLASHIAP (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLEXCAN (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLEXCOMM_I2C (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLEXIO_A_FORMAT (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLEXIO_CAMERA (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLEXIO_I2C (*C enumerator*), 355
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLEXIO_I2S (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLEXIO_MCULCD (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLEXIO_SPI (*C enumerator*), 355
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLEXIO_T_FORMAT (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLEXIO_UART (*C enumerator*), 355
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_FLEXSPI (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_Generic (*C enumerator*), 355
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_GLIKEY (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HAL_ADC_SENSOR (*C enumerator*), 361

drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HAL_FLASH (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HAL_GPIO (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HAL_I2C (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HAL_I2S (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HAL_PWM (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HAL_RNG (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HAL_SPI (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HAL_TIMER (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HAL_UART (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HASHCRYPT (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HIPERFACE (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_HOSTIF (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_I2C (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_I2S (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_I3C (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_I3CBUS (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_IAP (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_ICS (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_IPED (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_IUART (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_JPEGDEC (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LED (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LIST (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LMEM (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LOG (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LPC_I2C (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LPC_I2C_1 (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LPC_MINISPI (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LPC_QSPI (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LPC_SPI (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LPC_SPI_SSP (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LPC_USART (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LPI2C (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LPSCI (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LPSPI (*C enumerator*), 355
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LPUART (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_LTC (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_MCAN (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_MCG (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_MECC (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_MEM_MANAGER (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_MIPI_DSI (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_MMDC (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_MSCAN (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_MSG (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_NETC (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_NOTIFIER (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_NPX (*C enumerator*), 363
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_OSA (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_OTFAD (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_OTP (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_PDM (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_PHY (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_PNGDEC (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_POWER (*C enumerator*), 357

drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_POWER_MANAGER (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_PUF (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_QSCI (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_QSPI (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_QUEUEDSPI (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SAI (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SCG (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SDHC (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SDIF (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SDIOSLV (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SDK_FLEXSPINOR (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SDK_OCOTP (*C enumerator*), 362
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SDMA (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SDMMC (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SDRAMC (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SDSPI (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SEMA42 (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SEMC (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SERIALMANAGER (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SFA (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SHELL (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SMARTCARD (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SPC (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SPDIF (*C enumerator*), 359
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SPI (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_SPIFI (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_TIMERMANAGER (*C enumerator*), 361
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_TOUCH_PANEL (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_TRGMUX (*C enumerator*), 357
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_UART (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_USDHC (*C enumerator*), 358
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_VBAT (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_XRDC (*C enumerator*), 356
drivers_Kinetis_K_MK22F51212._status_groups.kStatusGroup_XSPI (*C enumerator*), 360
drivers_Kinetis_K_MK22F51212._uart_config (*C struct*), 540
drivers_Kinetis_K_MK22F51212._uart_config.baudRate_Bps (*C var*), 540
drivers_Kinetis_K_MK22F51212._uart_config.enableRx (*C var*), 540
drivers_Kinetis_K_MK22F51212._uart_config.enableRxRTS (*C var*), 540
drivers_Kinetis_K_MK22F51212._uart_config.enableTx (*C var*), 540
drivers_Kinetis_K_MK22F51212._uart_config.enableTxCTS (*C var*), 540
drivers_Kinetis_K_MK22F51212._uart_config.idleType (*C var*), 540
drivers_Kinetis_K_MK22F51212._uart_config.parityMode (*C var*), 540
drivers_Kinetis_K_MK22F51212._uart_config.rxFifoWatermark (*C var*), 540
drivers_Kinetis_K_MK22F51212._uart_config.stopBitCount (*C var*), 540
drivers_Kinetis_K_MK22F51212._uart_config.txFifoWatermark (*C var*), 540
drivers_Kinetis_K_MK22F51212._uart_edma_handle (*C struct*), 544
drivers_Kinetis_K_MK22F51212._uart_edma_handle.callback (*C var*), 544
drivers_Kinetis_K_MK22F51212._uart_edma_handle.nbytes (*C var*), 545
drivers_Kinetis_K_MK22F51212._uart_edma_handle.rxDataSizeAll (*C var*), 544
drivers_Kinetis_K_MK22F51212._uart_edma_handle.rxEdmaHandle (*C var*), 545
drivers_Kinetis_K_MK22F51212._uart_edma_handle.rxState (*C var*), 545
drivers_Kinetis_K_MK22F51212._uart_edma_handle.txDataSizeAll (*C var*), 544
drivers_Kinetis_K_MK22F51212._uart_edma_handle.txEdmaHandle (*C var*), 544

drivers_Kinetis_K_MK22F51212._uart_edma_handle.txState (*C var*), 545
drivers_Kinetis_K_MK22F51212._uart_edma_handle.userData (*C var*), 544
drivers_Kinetis_K_MK22F51212._uart_handle (*C struct*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.callback (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.rxData (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.rxDataSize (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.rxDataSizeAll (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.rxRingBuffer (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.rxRingBufferHead (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.rxRingBufferSize (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.rxRingBufferTail (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.rxState (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.txData (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.txDataSize (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.txDataSizeAll (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.txState (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_handle.userData (*C var*), 541
drivers_Kinetis_K_MK22F51212._uart_idle_type_select (*C enum*), 536
drivers_Kinetis_K_MK22F51212._uart_idle_type_select.kUART_IdleTypeStartBit (*C enumerator*), 536
drivers_Kinetis_K_MK22F51212._uart_idle_type_select.kUART_IdleTypeStopBit (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable (*C enum*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_AllInterruptsEnable (*C enumerator*), 538
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_FramingErrorInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_IdleLineInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_LinBreakInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_NoiseErrorInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_ParityErrorInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_RxActiveEdgeInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_RxDataRegFullInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_RxFifoOverflowInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_RxFifoUnderflowInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_RxOverrunInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_TransmissionCompleteInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_TxDataRegEmptyInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_interrupt_enable.kUART_TxFifoOverflowInterruptEnable (*C enumerator*), 537
drivers_Kinetis_K_MK22F51212._uart_parity_mode (*C enum*), 536
drivers_Kinetis_K_MK22F51212._uart_parity_mode.kUART_ParityDisabled (*C enumerator*), 536
drivers_Kinetis_K_MK22F51212._uart_parity_mode.kUART_ParityEven (*C enumerator*), 536
drivers_Kinetis_K_MK22F51212._uart_parity_mode.kUART_ParityOdd (*C enumerator*), 536
drivers_Kinetis_K_MK22F51212._uart_stop_bit_count (*C enum*), 536
drivers_Kinetis_K_MK22F51212._uart_stop_bit_count.kUART_OneStopBit (*C enumerator*), 536
drivers_Kinetis_K_MK22F51212._uart_stop_bit_count.kUART_TwoStopBit (*C enumerator*), 536

drivers_Kinetis_K_MK22F51212._uart_transfer (*C struct*), 540
drivers_Kinetis_K_MK22F51212._uart_transfer.___unnamed39___ (*C union*), 542
drivers_Kinetis_K_MK22F51212._uart_transfer.___unnamed39___.data (*C var*), 542
drivers_Kinetis_K_MK22F51212._uart_transfer.___unnamed39___.rxData (*C var*), 542
drivers_Kinetis_K_MK22F51212._uart_transfer.___unnamed39___.txData (*C var*), 542
drivers_Kinetis_K_MK22F51212._uart_transfer.dataSize (*C var*), 541
drivers_Kinetis_K_MK22F51212._vref_buffer_mode (*C enum*), 547
drivers_Kinetis_K_MK22F51212._vref_buffer_mode.kVREF_ModeBandgapOnly (*C enumerator*), 547
drivers_Kinetis_K_MK22F51212._vref_buffer_mode.kVREF_ModeHighPowerBuffer (*C enumerator*), 547
drivers_Kinetis_K_MK22F51212._vref_buffer_mode.kVREF_ModeLowPowerBuffer (*C enumerator*), 548
drivers_Kinetis_K_MK22F51212._vref_config (*C struct*), 549
drivers_Kinetis_K_MK22F51212._vref_config.bufferMode (*C var*), 549
drivers_Kinetis_K_MK22F51212._vref_config.enable2V1VoltRef (*C var*), 549
drivers_Kinetis_K_MK22F51212._vref_config.enableExternalVoltRef (*C var*), 549
drivers_Kinetis_K_MK22F51212._vref_config.enableLowRef (*C var*), 549
drivers_Kinetis_K_MK22F51212._wdog_clock_prescaler (*C enum*), 553
drivers_Kinetis_K_MK22F51212._wdog_clock_prescaler.kWDOG_ClockPrescalerDivide1 (*C enumerator*), 553
drivers_Kinetis_K_MK22F51212._wdog_clock_prescaler.kWDOG_ClockPrescalerDivide2 (*C enumerator*), 553
drivers_Kinetis_K_MK22F51212._wdog_clock_prescaler.kWDOG_ClockPrescalerDivide3 (*C enumerator*), 553
drivers_Kinetis_K_MK22F51212._wdog_clock_prescaler.kWDOG_ClockPrescalerDivide4 (*C enumerator*), 553
drivers_Kinetis_K_MK22F51212._wdog_clock_prescaler.kWDOG_ClockPrescalerDivide5 (*C enumerator*), 553
drivers_Kinetis_K_MK22F51212._wdog_clock_prescaler.kWDOG_ClockPrescalerDivide6 (*C enumerator*), 554
drivers_Kinetis_K_MK22F51212._wdog_clock_prescaler.kWDOG_ClockPrescalerDivide7 (*C enumerator*), 554
drivers_Kinetis_K_MK22F51212._wdog_clock_prescaler.kWDOG_ClockPrescalerDivide8 (*C enumerator*), 554
drivers_Kinetis_K_MK22F51212._wdog_clock_source (*C enum*), 553
drivers_Kinetis_K_MK22F51212._wdog_clock_source.kWDOG_AlternateClockSource (*C enumerator*), 553
drivers_Kinetis_K_MK22F51212._wdog_clock_source.kWDOG_LpoClockSource (*C enumerator*), 553
drivers_Kinetis_K_MK22F51212._wdog_config (*C struct*), 556
drivers_Kinetis_K_MK22F51212._wdog_config.clockSource (*C var*), 556
drivers_Kinetis_K_MK22F51212._wdog_config.enableInterrupt (*C var*), 556
drivers_Kinetis_K_MK22F51212._wdog_config.enableUpdate (*C var*), 556
drivers_Kinetis_K_MK22F51212._wdog_config.enableWdog (*C var*), 556
drivers_Kinetis_K_MK22F51212._wdog_config.enableWindowMode (*C var*), 556
drivers_Kinetis_K_MK22F51212._wdog_config.prescaler (*C var*), 556
drivers_Kinetis_K_MK22F51212._wdog_config.timeoutValue (*C var*), 556
drivers_Kinetis_K_MK22F51212._wdog_config.windowValue (*C var*), 556
drivers_Kinetis_K_MK22F51212._wdog_config.workMode (*C var*), 556
drivers_Kinetis_K_MK22F51212._wdog_interrupt_enable_t (*C enum*), 554
drivers_Kinetis_K_MK22F51212._wdog_interrupt_enable_t.kWDOG_InterruptEnable (*C enumerator*), 554
drivers_Kinetis_K_MK22F51212._wdog_status_flags_t (*C enum*), 554
drivers_Kinetis_K_MK22F51212._wdog_status_flags_t.kWDOG_RunningFlag (*C enumerator*), 555
drivers_Kinetis_K_MK22F51212._wdog_status_flags_t.kWDOG_TimeoutFlag (*C enumerator*), 555
drivers_Kinetis_K_MK22F51212._wdog_test_config (*C struct*), 556
drivers_Kinetis_K_MK22F51212._wdog_test_config.testedByte (*C var*), 556

drivers_Kinetis_K_MK22F51212._wdog_test_config.testMode (*C var*), 556
drivers_Kinetis_K_MK22F51212._wdog_test_config.timeoutValue (*C var*), 556
drivers_Kinetis_K_MK22F51212._wdog_test_mode (*C enum*), 554
drivers_Kinetis_K_MK22F51212._wdog_test_mode.kWDOG_ByteTest (*C enumerator*), 554
drivers_Kinetis_K_MK22F51212._wdog_test_mode.kWDOG_QuickTest (*C enumerator*), 554
drivers_Kinetis_K_MK22F51212._wdog_tested_byte (*C enum*), 554
drivers_Kinetis_K_MK22F51212._wdog_tested_byte.kWDOG_TestByte0 (*C enumerator*), 554
drivers_Kinetis_K_MK22F51212._wdog_tested_byte.kWDOG_TestByte1 (*C enumerator*), 554
drivers_Kinetis_K_MK22F51212._wdog_tested_byte.kWDOG_TestByte2 (*C enumerator*), 554
drivers_Kinetis_K_MK22F51212._wdog_tested_byte.kWDOG_TestByte3 (*C enumerator*), 554
drivers_Kinetis_K_MK22F51212._wdog_work_mode (*C struct*), 555
drivers_Kinetis_K_MK22F51212._wdog_work_mode.enableDebug (*C var*), 555
drivers_Kinetis_K_MK22F51212._wdog_work_mode.enableStop (*C var*), 555
drivers_Kinetis_K_MK22F51212._wdog_work_mode.enableWait (*C var*), 555
drivers_Kinetis_K_MK22F51212.adc16_channel_config_t (*C type*), 147
drivers_Kinetis_K_MK22F51212.adc16_channel_mux_mode_t (*C type*), 146
drivers_Kinetis_K_MK22F51212.ADC16_ClearStatusFlags (*C function*), 141
drivers_Kinetis_K_MK22F51212.adc16_clock_divider_t (*C type*), 146
drivers_Kinetis_K_MK22F51212.adc16_clock_source_t (*C type*), 146
drivers_Kinetis_K_MK22F51212.ADC16_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.adc16_config_t (*C type*), 147
drivers_Kinetis_K_MK22F51212.ADC16_Deinit (*C function*), 139
drivers_Kinetis_K_MK22F51212.ADC16_DoAutoCalibration (*C function*), 139
drivers_Kinetis_K_MK22F51212.ADC16_EnableAsynchronousClockOutput (*C function*), 141
drivers_Kinetis_K_MK22F51212.ADC16_EnableDMA (*C function*), 140
drivers_Kinetis_K_MK22F51212.ADC16_EnableHardwareTrigger (*C function*), 140
drivers_Kinetis_K_MK22F51212.ADC16_GetChannelConversionValue (*C function*), 142
drivers_Kinetis_K_MK22F51212.ADC16_GetChannelStatusFlags (*C function*), 142
drivers_Kinetis_K_MK22F51212.ADC16_GetDefaultConfig (*C function*), 139
drivers_Kinetis_K_MK22F51212.ADC16_GetStatusFlags (*C function*), 141
drivers_Kinetis_K_MK22F51212.adc16_hardware_average_mode_t (*C type*), 147
drivers_Kinetis_K_MK22F51212.adc16_hardware_compare_config_t (*C type*), 147
drivers_Kinetis_K_MK22F51212.adc16_hardware_compare_mode_t (*C type*), 147
drivers_Kinetis_K_MK22F51212.ADC16_Init (*C function*), 139
drivers_Kinetis_K_MK22F51212.adc16_long_sample_mode_t (*C type*), 146
drivers_Kinetis_K_MK22F51212.adc16_pga_config_t (*C type*), 147
drivers_Kinetis_K_MK22F51212.adc16_pga_gain_t (*C type*), 147
drivers_Kinetis_K_MK22F51212.adc16_reference_voltage_source_t (*C type*), 147
drivers_Kinetis_K_MK22F51212.adc16_resolution_t (*C type*), 146
drivers_Kinetis_K_MK22F51212.ADC16_SetChannelConfig (*C function*), 141
drivers_Kinetis_K_MK22F51212.ADC16_SetChannelMuxMode (*C function*), 140
drivers_Kinetis_K_MK22F51212.ADC16_SetHardwareAverage (*C function*), 141
drivers_Kinetis_K_MK22F51212.ADC16_SetHardwareCompareConfig (*C function*), 140
drivers_Kinetis_K_MK22F51212.ADC16_SetOffsetValue (*C function*), 140
drivers_Kinetis_K_MK22F51212.ADC16_SetPGAConfig (*C function*), 141
drivers_Kinetis_K_MK22F51212.ALIGN_DOWN (*C macro*), 303
drivers_Kinetis_K_MK22F51212.ALIGN_UP (*C macro*), 303
drivers_Kinetis_K_MK22F51212.ARRAY_SIZE (*C macro*), 367
drivers_Kinetis_K_MK22F51212.AT_CACHE_LINE_SECTION (*C macro*), 354
drivers_Kinetis_K_MK22F51212.AT_CACHE_LINE_SECTION_INIT (*C macro*), 354
drivers_Kinetis_K_MK22F51212.AT_NONCACHEABLE_SECTION (*C macro*), 354
drivers_Kinetis_K_MK22F51212.AT_NONCACHEABLE_SECTION_ALIGN (*C macro*), 354
drivers_Kinetis_K_MK22F51212.AT_NONCACHEABLE_SECTION_ALIGN_INIT (*C macro*), 354
drivers_Kinetis_K_MK22F51212.AT_NONCACHEABLE_SECTION_INIT (*C macro*), 354
drivers_Kinetis_K_MK22F51212.AT_QUICKACCESS_SECTION_CODE (*C macro*), 354
drivers_Kinetis_K_MK22F51212.AT_QUICKACCESS_SECTION_DATA (*C macro*), 355
drivers_Kinetis_K_MK22F51212.AT_QUICKACCESS_SECTION_DATA_ALIGN (*C macro*), 355
drivers_Kinetis_K_MK22F51212.AUTOBAUD_BAUDRATE_TOLERANCE (*C macro*), 375

drivers_Kinetis_K_MK22F51212.AUTOBAUD_BREAK_TIME_MIN (*C macro*), 376
drivers_Kinetis_K_MK22F51212.B1P1 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.B1P2 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.B1P3 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.B1P4 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.B2P1 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.B2P2 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.B2P3 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.B3P1 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.B3P2 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.BIT_DURATION_MAX_2400 (*C macro*), 376
drivers_Kinetis_K_MK22F51212.BIT_DURATION_MAX_4800 (*C macro*), 375
drivers_Kinetis_K_MK22F51212.BIT_DURATION_MAX_9600 (*C macro*), 375
drivers_Kinetis_K_MK22F51212.BIT_DURATION_MAX_14400 (*C macro*), 375
drivers_Kinetis_K_MK22F51212.BIT_DURATION_MAX_19200 (*C macro*), 375
drivers_Kinetis_K_MK22F51212.BIT_DURATION_MIN_2400 (*C macro*), 376
drivers_Kinetis_K_MK22F51212.BIT_DURATION_MIN_4800 (*C macro*), 375
drivers_Kinetis_K_MK22F51212.BIT_DURATION_MIN_9600 (*C macro*), 375
drivers_Kinetis_K_MK22F51212.BIT_DURATION_MIN_14400 (*C macro*), 375
drivers_Kinetis_K_MK22F51212.BIT_DURATION_MIN_19200 (*C macro*), 375
drivers_Kinetis_K_MK22F51212.BIT_RATE_TOLERANCE_UNSYNC (*C macro*), 375
drivers_Kinetis_K_MK22F51212.BRFA_CAL_ADJUST_D1_T0 (*C macro*), 517
drivers_Kinetis_K_MK22F51212.BRFA_CAL_ADJUST_D1_T1 (*C macro*), 517
drivers_Kinetis_K_MK22F51212.BRFA_CAL_ADJUST_D2_T0 (*C macro*), 517
drivers_Kinetis_K_MK22F51212.BRFA_CAL_ADJUST_D2_T1 (*C macro*), 518
drivers_Kinetis_K_MK22F51212.BRFA_CAL_ADJUST_D4_T0 (*C macro*), 517
drivers_Kinetis_K_MK22F51212.BRFA_CAL_ADJUST_D4_T1 (*C macro*), 518
drivers_Kinetis_K_MK22F51212.BUS_CLK (*C macro*), 164
drivers_Kinetis_K_MK22F51212.BYTE2WORD_1_1_1_1 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.BYTE2WORD_1_1_2 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.BYTE2WORD_1_2_1 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.BYTE2WORD_1_3 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.BYTE2WORD_2_1_1 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.BYTE2WORD_2_2 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.BYTE2WORD_3_1 (*C macro*), 303
drivers_Kinetis_K_MK22F51212.capLoad (*C var*), 175
drivers_Kinetis_K_MK22F51212.CLK_GATE_ABSTRACT_BITS_SHIFT (*C macro*), 164
drivers_Kinetis_K_MK22F51212.CLK_GATE_ABSTRACT_REG_OFFSET (*C macro*), 164
drivers_Kinetis_K_MK22F51212.CLK_GATE_BIT_SHIFT_MASK (*C macro*), 164
drivers_Kinetis_K_MK22F51212.CLK_GATE_BIT_SHIFT_SHIFT (*C macro*), 164
drivers_Kinetis_K_MK22F51212.CLK_GATE_DEFINE (*C macro*), 164
drivers_Kinetis_K_MK22F51212.CLK_GATE_REG_OFFSET_MASK (*C macro*), 164
drivers_Kinetis_K_MK22F51212.CLK_GATE_REG_OFFSET_SHIFT (*C macro*), 164
drivers_Kinetis_K_MK22F51212.clkdiv1 (*C var*), 174
drivers_Kinetis_K_MK22F51212.CLOCK_BootToBlpeMode (*C function*), 174
drivers_Kinetis_K_MK22F51212.CLOCK_BootToBlpiMode (*C function*), 173
drivers_Kinetis_K_MK22F51212.CLOCK_BootToFeeMode (*C function*), 173
drivers_Kinetis_K_MK22F51212.CLOCK_BootToFeiMode (*C function*), 172
drivers_Kinetis_K_MK22F51212.CLOCK_BootToPeeMode (*C function*), 174
drivers_Kinetis_K_MK22F51212.CLOCK_CalcPllDiv (*C function*), 166
drivers_Kinetis_K_MK22F51212.CLOCK_ClearStatusFlags (*C function*), 167
drivers_Kinetis_K_MK22F51212.CLOCK_DeinitOsc0 (*C function*), 168
drivers_Kinetis_K_MK22F51212.CLOCK_DisableClock (*C function*), 159
drivers_Kinetis_K_MK22F51212.CLOCK_DisablePll0 (*C function*), 166
drivers_Kinetis_K_MK22F51212.CLOCK_DisableUsbfs0Clock (*C function*), 160
drivers_Kinetis_K_MK22F51212.CLOCK_EnableClock (*C function*), 159
drivers_Kinetis_K_MK22F51212.CLOCK_EnablePll0 (*C function*), 166
drivers_Kinetis_K_MK22F51212.CLOCK_EnableUsbfs0Clock (*C function*), 160

drivers_Kinetis_K_MK22F51212.CLOCK_ExternalModeToFbeModeQuick (*C function*), 172
drivers_Kinetis_K_MK22F51212.CLOCK_GetBusClkFreq (*C function*), 161
drivers_Kinetis_K_MK22F51212.CLOCK_GetCoreSysClkFreq (*C function*), 161
drivers_Kinetis_K_MK22F51212.CLOCK_GetEr32kClkFreq (*C function*), 161
drivers_Kinetis_K_MK22F51212.CLOCK_GetFixedFreqClkFreq (*C function*), 165
drivers_Kinetis_K_MK22F51212.CLOCK_GetFlashClkFreq (*C function*), 161
drivers_Kinetis_K_MK22F51212.CLOCK_GetFlexBusClkFreq (*C function*), 161
drivers_Kinetis_K_MK22F51212.CLOCK_GetFllFreq (*C function*), 164
drivers_Kinetis_K_MK22F51212.CLOCK_GetFreq (*C function*), 161
drivers_Kinetis_K_MK22F51212.CLOCK_GetInternalRefClkFreq (*C function*), 165
drivers_Kinetis_K_MK22F51212.CLOCK_GetMode (*C function*), 169
drivers_Kinetis_K_MK22F51212.CLOCK_GetOsc0ErClkDivFreq (*C function*), 162
drivers_Kinetis_K_MK22F51212.CLOCK_GetOsc0ErClkFreq (*C function*), 162
drivers_Kinetis_K_MK22F51212.CLOCK_GetOsc0ErClkUndivFreq (*C function*), 161
drivers_Kinetis_K_MK22F51212.CLOCK_GetOutClkFreq (*C function*), 164
drivers_Kinetis_K_MK22F51212.CLOCK_GetPlatClkFreq (*C function*), 161
drivers_Kinetis_K_MK22F51212.CLOCK_GetPll0Freq (*C function*), 165
drivers_Kinetis_K_MK22F51212.CLOCK_GetPllFllSelClkFreq (*C function*), 161
drivers_Kinetis_K_MK22F51212.CLOCK_GetStatusFlags (*C function*), 167
drivers_Kinetis_K_MK22F51212.CLOCK_InitOsc0 (*C function*), 168
drivers_Kinetis_K_MK22F51212.CLOCK_InternalModeToFbiModeQuick (*C function*), 172
drivers_Kinetis_K_MK22F51212.clock_ip_name_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.clock_name_t (*C type*), 157
drivers_Kinetis_K_MK22F51212.CLOCK_SetBlpeMode (*C function*), 171
drivers_Kinetis_K_MK22F51212.CLOCK_SetBlpiMode (*C function*), 171
drivers_Kinetis_K_MK22F51212.CLOCK_SetClkOutClock (*C function*), 160
drivers_Kinetis_K_MK22F51212.CLOCK_SetEr32kClock (*C function*), 160
drivers_Kinetis_K_MK22F51212.CLOCK_SetExternalRefClkConfig (*C function*), 166
drivers_Kinetis_K_MK22F51212.CLOCK_SetFastIrcFreq (*C function*), 169
drivers_Kinetis_K_MK22F51212.CLOCK_SetFbeMode (*C function*), 171
drivers_Kinetis_K_MK22F51212.CLOCK_SetFbiMode (*C function*), 170
drivers_Kinetis_K_MK22F51212.CLOCK_SetFeeMode (*C function*), 170
drivers_Kinetis_K_MK22F51212.CLOCK_SetFeiMode (*C function*), 169
drivers_Kinetis_K_MK22F51212.CLOCK_SetFllExtRefDiv (*C function*), 166
drivers_Kinetis_K_MK22F51212.CLOCK_SetInternalRefClkConfig (*C function*), 165
drivers_Kinetis_K_MK22F51212.CLOCK_SetLowPowerEnable (*C function*), 165
drivers_Kinetis_K_MK22F51212.CLOCK_SetLpuartClock (*C function*), 160
drivers_Kinetis_K_MK22F51212.CLOCK_SetMcgConfig (*C function*), 174
drivers_Kinetis_K_MK22F51212.CLOCK_SetOsc0MonitorMode (*C function*), 167
drivers_Kinetis_K_MK22F51212.CLOCK_SetOutDiv (*C function*), 160
drivers_Kinetis_K_MK22F51212.CLOCK_SetPbeMode (*C function*), 171
drivers_Kinetis_K_MK22F51212.CLOCK_SetPeeMode (*C function*), 172
drivers_Kinetis_K_MK22F51212.CLOCK_SetPll0MonitorMode (*C function*), 167
drivers_Kinetis_K_MK22F51212.CLOCK_SetPllFllSelClock (*C function*), 160
drivers_Kinetis_K_MK22F51212.CLOCK_SetRtcClkOutClock (*C function*), 160
drivers_Kinetis_K_MK22F51212.CLOCK_SetRtcOscMonitorMode (*C function*), 167
drivers_Kinetis_K_MK22F51212.CLOCK_SetSimConfig (*C function*), 162
drivers_Kinetis_K_MK22F51212.CLOCK_SetSimSafeDivs (*C function*), 162
drivers_Kinetis_K_MK22F51212.CLOCK_SetSlowIrcFreq (*C function*), 169
drivers_Kinetis_K_MK22F51212.CLOCK_SetTraceClock (*C function*), 160
drivers_Kinetis_K_MK22F51212.CLOCK_SetXtal0Freq (*C function*), 168
drivers_Kinetis_K_MK22F51212.CLOCK_SetXtal32Freq (*C function*), 168
drivers_Kinetis_K_MK22F51212.CLOCK_TrimInternalRefClk (*C function*), 169
drivers_Kinetis_K_MK22F51212.clock_usb_src_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.CMP_ClearStatusFlags (*C function*), 179
drivers_Kinetis_K_MK22F51212.CMP_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.cmp_config_t (*C type*), 180
drivers_Kinetis_K_MK22F51212.cmp_dac_config_t (*C type*), 180

drivers_Kinetis_K_MK22F51212.CMP_Deinit (*C function*), 177
drivers_Kinetis_K_MK22F51212.CMP_DisableInterrupts (*C function*), 179
drivers_Kinetis_K_MK22F51212.CMP_Enable (*C function*), 177
drivers_Kinetis_K_MK22F51212.CMP_EnableDMA (*C function*), 178
drivers_Kinetis_K_MK22F51212.CMP_EnableInterrupts (*C function*), 179
drivers_Kinetis_K_MK22F51212.CMP_EnablePassThroughMode (*C function*), 178
drivers_Kinetis_K_MK22F51212.CMP_EnableWindowMode (*C function*), 178
drivers_Kinetis_K_MK22F51212.cmp_filter_config_t (*C type*), 180
drivers_Kinetis_K_MK22F51212.CMP_GetDefaultConfig (*C function*), 177
drivers_Kinetis_K_MK22F51212.CMP_GetStatusFlags (*C function*), 179
drivers_Kinetis_K_MK22F51212.cmp_hysteresis_mode_t (*C type*), 180
drivers_Kinetis_K_MK22F51212.CMP_Init (*C function*), 177
drivers_Kinetis_K_MK22F51212.cmp_reference_voltage_source_t (*C type*), 180
drivers_Kinetis_K_MK22F51212.CMP_SetDACConfig (*C function*), 178
drivers_Kinetis_K_MK22F51212.CMP_SetFilterConfig (*C function*), 178
drivers_Kinetis_K_MK22F51212.CMP_SetInputChannels (*C function*), 178
drivers_Kinetis_K_MK22F51212.COUNT_TO_MSEC (*C macro*), 354
drivers_Kinetis_K_MK22F51212.COUNT_TO_USEC (*C macro*), 354
drivers_Kinetis_K_MK22F51212.crc_bits_t (*C type*), 183
drivers_Kinetis_K_MK22F51212.CRC_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.crc_config_t (*C type*), 183
drivers_Kinetis_K_MK22F51212.CRC_Deinit (*C function*), 183
drivers_Kinetis_K_MK22F51212.CRC_DRIVER_USE_CRC16_CCIT_FALSE_AS_DEFAULT (*C macro*), 184
drivers_Kinetis_K_MK22F51212.CRC_Get16bitResult (*C function*), 184
drivers_Kinetis_K_MK22F51212.CRC_Get32bitResult (*C function*), 184
drivers_Kinetis_K_MK22F51212.CRC_GetDefaultConfig (*C function*), 183
drivers_Kinetis_K_MK22F51212.CRC_Init (*C function*), 183
drivers_Kinetis_K_MK22F51212.crc_result_t (*C type*), 183
drivers_Kinetis_K_MK22F51212.CRC_WriteData (*C function*), 183
drivers_Kinetis_K_MK22F51212.dac_buffer_config_t (*C type*), 190
drivers_Kinetis_K_MK22F51212.dac_buffer_trigger_mode_t (*C type*), 189
drivers_Kinetis_K_MK22F51212.dac_buffer_watermark_t (*C type*), 189
drivers_Kinetis_K_MK22F51212.dac_buffer_work_mode_t (*C type*), 189
drivers_Kinetis_K_MK22F51212.DAC_ClearBufferStatusFlags (*C function*), 187
drivers_Kinetis_K_MK22F51212.DAC_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.dac_config_t (*C type*), 189
drivers_Kinetis_K_MK22F51212.DAC_Deinit (*C function*), 185
drivers_Kinetis_K_MK22F51212.DAC_DisableBufferInterrupts (*C function*), 187
drivers_Kinetis_K_MK22F51212.DAC_DoSoftwareTriggerBuffer (*C function*), 186
drivers_Kinetis_K_MK22F51212.DAC_Enable (*C function*), 186
drivers_Kinetis_K_MK22F51212.DAC_EnableBuffer (*C function*), 186
drivers_Kinetis_K_MK22F51212.DAC_EnableBufferDMA (*C function*), 186
drivers_Kinetis_K_MK22F51212.DAC_EnableBufferInterrupts (*C function*), 187
drivers_Kinetis_K_MK22F51212.DAC_GetBufferReadPointer (*C function*), 187
drivers_Kinetis_K_MK22F51212.DAC_GetBufferStatusFlags (*C function*), 187
drivers_Kinetis_K_MK22F51212.DAC_GetDefaultBufferConfig (*C function*), 186
drivers_Kinetis_K_MK22F51212.DAC_GetDefaultConfig (*C function*), 185
drivers_Kinetis_K_MK22F51212.DAC_Init (*C function*), 185
drivers_Kinetis_K_MK22F51212.dac_reference_voltage_source_t (*C type*), 189
drivers_Kinetis_K_MK22F51212.DAC_SetBufferConfig (*C function*), 186
drivers_Kinetis_K_MK22F51212.DAC_SetBufferReadPointer (*C function*), 187
drivers_Kinetis_K_MK22F51212.DAC_SetBufferValue (*C function*), 186
drivers_Kinetis_K_MK22F51212.DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM (*C macro*), 351
drivers_Kinetis_K_MK22F51212.DEBUG_CONSOLE_DEVICE_TYPE_IUART (*C macro*), 351
drivers_Kinetis_K_MK22F51212.DEBUG_CONSOLE_DEVICE_TYPE_LPSCI (*C macro*), 351
drivers_Kinetis_K_MK22F51212.DEBUG_CONSOLE_DEVICE_TYPE_LPUART (*C macro*), 351

drivers_Kinetis_K_MK22F51212.DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART (*C macro*), 352
drivers_Kinetis_K_MK22F51212.DEBUG_CONSOLE_DEVICE_TYPE_NONE (*C macro*), 351
drivers_Kinetis_K_MK22F51212.DEBUG_CONSOLE_DEVICE_TYPE_QSCI (*C macro*), 352
drivers_Kinetis_K_MK22F51212.DEBUG_CONSOLE_DEVICE_TYPE_SWO (*C macro*), 352
drivers_Kinetis_K_MK22F51212.DEBUG_CONSOLE_DEVICE_TYPE_UART (*C macro*), 351
drivers_Kinetis_K_MK22F51212.DEBUG_CONSOLE_DEVICE_TYPE_USBCDC (*C macro*), 351
drivers_Kinetis_K_MK22F51212.DEBUG_CONSOLE_DEVICE_TYPE_VUSART (*C macro*), 352
drivers_Kinetis_K_MK22F51212.DisableGlobalIRQ (*C function*), 366
drivers_Kinetis_K_MK22F51212.DisableIRQ (*C function*), 365
drivers_Kinetis_K_MK22F51212.DMA_DCHPRI_INDEX (*C macro*), 243
drivers_Kinetis_K_MK22F51212.DMAMUX_CLOCKS (*C macro*), 162
drivers_Kinetis_K_MK22F51212.DMAMUX_Deinit (*C function*), 190
drivers_Kinetis_K_MK22F51212.DMAMUX_DisableChannel (*C function*), 191
drivers_Kinetis_K_MK22F51212.DMAMUX_DisablePeriodTrigger (*C function*), 191
drivers_Kinetis_K_MK22F51212.DMAMUX_EnableChannel (*C function*), 191
drivers_Kinetis_K_MK22F51212.DMAMUX_EnablePeriodTrigger (*C function*), 191
drivers_Kinetis_K_MK22F51212.DMAMUX_Init (*C function*), 190
drivers_Kinetis_K_MK22F51212.DMAMUX_SetSource (*C function*), 191
drivers_Kinetis_K_MK22F51212.dmx32 (*C var*), 175
drivers_Kinetis_K_MK22F51212.drs (*C var*), 175
drivers_Kinetis_K_MK22F51212.DSPI0_CLK_SRC (*C macro*), 164
drivers_Kinetis_K_MK22F51212.DSPI1_CLK_SRC (*C macro*), 164
drivers_Kinetis_K_MK22F51212.DSPI_ClearStatusFlags (*C function*), 193
drivers_Kinetis_K_MK22F51212.dspi_clock_phase_t (*C type*), 211
drivers_Kinetis_K_MK22F51212.dspi_clock_polarity_t (*C type*), 211
drivers_Kinetis_K_MK22F51212.DSPI_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.dspi_command_data_config_t (*C type*), 211
drivers_Kinetis_K_MK22F51212.dspi_ctar_selection_t (*C type*), 211
drivers_Kinetis_K_MK22F51212.DSPI_Deinit (*C function*), 193
drivers_Kinetis_K_MK22F51212.dspi_delay_type_t (*C type*), 211
drivers_Kinetis_K_MK22F51212.DSPI_DisableDMA (*C function*), 194
drivers_Kinetis_K_MK22F51212.DSPI_DisableInterrupts (*C function*), 194
drivers_Kinetis_K_MK22F51212.DSPI_DriverIRQHandler (*C function*), 204
drivers_Kinetis_K_MK22F51212.DSPI_DUMMY_DATA (*C macro*), 213
drivers_Kinetis_K_MK22F51212.DSPI_EDMA_MAX_TRANSFER_SIZE (*C macro*), 223
drivers_Kinetis_K_MK22F51212.DSPI_Enable (*C function*), 193
drivers_Kinetis_K_MK22F51212.DSPI_EnableDMA (*C function*), 194
drivers_Kinetis_K_MK22F51212.DSPI_EnableInterrupts (*C function*), 194
drivers_Kinetis_K_MK22F51212.DSPI_FlushFifo (*C function*), 196
drivers_Kinetis_K_MK22F51212.DSPI_GetDefaultDataCommandConfig (*C function*), 198
drivers_Kinetis_K_MK22F51212.DSPI_GetDummyDataInstance (*C function*), 204
drivers_Kinetis_K_MK22F51212.DSPI_GetInstance (*C function*), 195
drivers_Kinetis_K_MK22F51212.DSPI_GetRxRegisterAddress (*C function*), 195
drivers_Kinetis_K_MK22F51212.DSPI_GetStatusFlags (*C function*), 193
drivers_Kinetis_K_MK22F51212.dspi_half_duplex_transfer_t (*C type*), 213
drivers_Kinetis_K_MK22F51212.DSPI_IsMaster (*C function*), 195
drivers_Kinetis_K_MK22F51212.dspi_master_config_t (*C type*), 212
drivers_Kinetis_K_MK22F51212.dspi_master_ctar_config_t (*C type*), 212
drivers_Kinetis_K_MK22F51212.DSPI_MASTER_CTAR_MASK (*C macro*), 213
drivers_Kinetis_K_MK22F51212.DSPI_MASTER_CTAR_SHIFT (*C macro*), 213
drivers_Kinetis_K_MK22F51212.dspi_master_edma_handle_t (*C type*), 222
drivers_Kinetis_K_MK22F51212.dspi_master_edma_transfer_callback_t (*C type*), 222
drivers_Kinetis_K_MK22F51212.dspi_master_handle_t (*C type*), 212
drivers_Kinetis_K_MK22F51212.DSPI_MASTER_PCS_MASK (*C macro*), 213
drivers_Kinetis_K_MK22F51212.DSPI_MASTER_PCS_SHIFT (*C macro*), 213
drivers_Kinetis_K_MK22F51212.dspi_master_sample_point_t (*C type*), 211
drivers_Kinetis_K_MK22F51212.dspi_master_slave_mode_t (*C type*), 211

drivers_Kinetis_K_MK22F51212.dspi_master_transfer_callback_t (*C type*), 212
drivers_Kinetis_K_MK22F51212.DSPI_MasterGetDefaultConfig (*C function*), 192
drivers_Kinetis_K_MK22F51212.DSPI_MasterGetFormattedCommand (*C function*), 199
drivers_Kinetis_K_MK22F51212.DSPI_MasterGetTxRegisterAddress (*C function*), 195
drivers_Kinetis_K_MK22F51212.DSPI_MasterHalfDuplexTransferBlocking (*C function*), 202
drivers_Kinetis_K_MK22F51212.DSPI_MasterHalfDuplexTransferEDMA (*C function*), 220
drivers_Kinetis_K_MK22F51212.DSPI_MasterHalfDuplexTransferNonBlocking (*C function*), 202
drivers_Kinetis_K_MK22F51212.DSPI_MasterInit (*C function*), 192
drivers_Kinetis_K_MK22F51212.DSPI_MasterSetBaudRate (*C function*), 197
drivers_Kinetis_K_MK22F51212.DSPI_MasterSetDelayScaler (*C function*), 197
drivers_Kinetis_K_MK22F51212.DSPI_MasterSetDelayTimes (*C function*), 197
drivers_Kinetis_K_MK22F51212.DSPI_MasterTransferAbort (*C function*), 203
drivers_Kinetis_K_MK22F51212.DSPI_MasterTransferAbortEDMA (*C function*), 220
drivers_Kinetis_K_MK22F51212.DSPI_MasterTransferBlocking (*C function*), 201
drivers_Kinetis_K_MK22F51212.DSPI_MasterTransferCreateHandle (*C function*), 201
drivers_Kinetis_K_MK22F51212.DSPI_MasterTransferCreateHandleEDMA (*C function*), 219
drivers_Kinetis_K_MK22F51212.DSPI_MasterTransferEDMA (*C function*), 219
drivers_Kinetis_K_MK22F51212.DSPI_MasterTransferGetCount (*C function*), 202
drivers_Kinetis_K_MK22F51212.DSPI_MasterTransferGetCountEDMA (*C function*), 220
drivers_Kinetis_K_MK22F51212.DSPI_MasterTransferHandleIRQ (*C function*), 203
drivers_Kinetis_K_MK22F51212.DSPI_MasterTransferNonBlocking (*C function*), 201
drivers_Kinetis_K_MK22F51212.DSPI_MasterWriteCommandDataBlocking (*C function*), 199
drivers_Kinetis_K_MK22F51212.DSPI_MasterWriteData (*C function*), 198
drivers_Kinetis_K_MK22F51212.DSPI_MasterWriteDataBlocking (*C function*), 199
drivers_Kinetis_K_MK22F51212.dspi_pcs_polarity_config_t (*C type*), 211
drivers_Kinetis_K_MK22F51212.DSPI_ReadData (*C function*), 201
drivers_Kinetis_K_MK22F51212.DSPI_SetAllPcsPolarity (*C function*), 196
drivers_Kinetis_K_MK22F51212.DSPI_SetDummyData (*C function*), 201
drivers_Kinetis_K_MK22F51212.DSPI_SetFifoEnable (*C function*), 196
drivers_Kinetis_K_MK22F51212.DSPI_SetMasterSlaveMode (*C function*), 195
drivers_Kinetis_K_MK22F51212.dspi_shift_direction_t (*C type*), 211
drivers_Kinetis_K_MK22F51212.dspi_slave_config_t (*C type*), 212
drivers_Kinetis_K_MK22F51212.dspi_slave_ctar_config_t (*C type*), 212
drivers_Kinetis_K_MK22F51212.DSPI_SLAVE_CTAR_MASK (*C macro*), 213
drivers_Kinetis_K_MK22F51212.DSPI_SLAVE_CTAR_SHIFT (*C macro*), 213
drivers_Kinetis_K_MK22F51212.dspi_slave_edma_handle_t (*C type*), 222
drivers_Kinetis_K_MK22F51212.dspi_slave_edma_transfer_callback_t (*C type*), 222
drivers_Kinetis_K_MK22F51212.dspi_slave_handle_t (*C type*), 212
drivers_Kinetis_K_MK22F51212.dspi_slave_transfer_callback_t (*C type*), 212
drivers_Kinetis_K_MK22F51212.DSPI_SlaveGetDefaultConfig (*C function*), 193
drivers_Kinetis_K_MK22F51212.DSPI_SlaveGetTxRegisterAddress (*C function*), 195
drivers_Kinetis_K_MK22F51212.DSPI_SlaveInit (*C function*), 192
drivers_Kinetis_K_MK22F51212.DSPI_SlaveTransferAbort (*C function*), 204
drivers_Kinetis_K_MK22F51212.DSPI_SlaveTransferAbortEDMA (*C function*), 221
drivers_Kinetis_K_MK22F51212.DSPI_SlaveTransferCreateHandle (*C function*), 203
drivers_Kinetis_K_MK22F51212.DSPI_SlaveTransferCreateHandleEDMA (*C function*), 220
drivers_Kinetis_K_MK22F51212.DSPI_SlaveTransferEDMA (*C function*), 221
drivers_Kinetis_K_MK22F51212.DSPI_SlaveTransferGetCount (*C function*), 203
drivers_Kinetis_K_MK22F51212.DSPI_SlaveTransferGetCountEDMA (*C function*), 222
drivers_Kinetis_K_MK22F51212.DSPI_SlaveTransferHandleIRQ (*C function*), 204
drivers_Kinetis_K_MK22F51212.DSPI_SlaveTransferNonBlocking (*C function*), 203
drivers_Kinetis_K_MK22F51212.DSPI_SlaveWriteData (*C function*), 200
drivers_Kinetis_K_MK22F51212.DSPI_SlaveWriteDataBlocking (*C function*), 201
drivers_Kinetis_K_MK22F51212.DSPI_StartTransfer (*C function*), 196
drivers_Kinetis_K_MK22F51212.DSPI_StopTransfer (*C function*), 196
drivers_Kinetis_K_MK22F51212.dspi_transfer_t (*C type*), 212
drivers_Kinetis_K_MK22F51212.dspi_which_pcs_t (*C type*), 211
drivers_Kinetis_K_MK22F51212.EDMA_AbortTransfer (*C function*), 236

drivers_Kinetis_K_MK22F51212.edma_bandwidth_t (*C type*), 242
drivers_Kinetis_K_MK22F51212.edma_callback (*C type*), 243
drivers_Kinetis_K_MK22F51212.edma_channel_link_type_t (*C type*), 242
drivers_Kinetis_K_MK22F51212.edma_channel_Preemption_config_t (*C type*), 243
drivers_Kinetis_K_MK22F51212.EDMA_ClearChannelStatusFlags (*C function*), 234
drivers_Kinetis_K_MK22F51212.EDMA_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.edma_config_t (*C type*), 243
drivers_Kinetis_K_MK22F51212.EDMA_CreateHandle (*C function*), 234
drivers_Kinetis_K_MK22F51212.EDMA_Deinit (*C function*), 226
drivers_Kinetis_K_MK22F51212.EDMA_DisableChannelInterrupts (*C function*), 229
drivers_Kinetis_K_MK22F51212.EDMA_DisableChannelRequest (*C function*), 232
drivers_Kinetis_K_MK22F51212.EDMA_DriverIRQHandler (*C function*), 237
drivers_Kinetis_K_MK22F51212.EDMA_EnableAsyncRequest (*C function*), 229
drivers_Kinetis_K_MK22F51212.EDMA_EnableAutoStopRequest (*C function*), 229
drivers_Kinetis_K_MK22F51212.EDMA_EnableChannelInterrupts (*C function*), 229
drivers_Kinetis_K_MK22F51212.EDMA_EnableChannelRequest (*C function*), 232
drivers_Kinetis_K_MK22F51212.EDMA_EnableContinuousChannelLinkMode (*C function*), 226
drivers_Kinetis_K_MK22F51212.EDMA_EnableMinorLoopMapping (*C function*), 226
drivers_Kinetis_K_MK22F51212.EDMA_GetChannelStatusFlags (*C function*), 233
drivers_Kinetis_K_MK22F51212.EDMA_GetDefaultConfig (*C function*), 226
drivers_Kinetis_K_MK22F51212.EDMA_GetErrorStatusFlags (*C function*), 233
drivers_Kinetis_K_MK22F51212.EDMA_GetNextTCDAddress (*C function*), 236
drivers_Kinetis_K_MK22F51212.EDMA_GetRemainingMajorLoopCount (*C function*), 233
drivers_Kinetis_K_MK22F51212.EDMA_GetUnusedTCDNumber (*C function*), 236
drivers_Kinetis_K_MK22F51212.edma_handle_t (*C type*), 243
drivers_Kinetis_K_MK22F51212.EDMA_HandleIRQ (*C function*), 236
drivers_Kinetis_K_MK22F51212.EDMA_Init (*C function*), 225
drivers_Kinetis_K_MK22F51212.EDMA_InstallTCD (*C function*), 226
drivers_Kinetis_K_MK22F51212.EDMA_InstallTCDMemory (*C function*), 234
drivers_Kinetis_K_MK22F51212.edma_interrupt_enable_t (*C type*), 242
drivers_Kinetis_K_MK22F51212.edma_minor_offset_config_t (*C type*), 243
drivers_Kinetis_K_MK22F51212.edma_modulo_t (*C type*), 242
drivers_Kinetis_K_MK22F51212.EDMA_PrepareTransfer (*C function*), 235
drivers_Kinetis_K_MK22F51212.EDMA_PrepareTransferConfig (*C function*), 234
drivers_Kinetis_K_MK22F51212.EDMA_ResetChannel (*C function*), 226
drivers_Kinetis_K_MK22F51212.EDMA_SetBandWidth (*C function*), 228
drivers_Kinetis_K_MK22F51212.EDMA_SetCallback (*C function*), 234
drivers_Kinetis_K_MK22F51212.EDMA_SetChannelLink (*C function*), 228
drivers_Kinetis_K_MK22F51212.EDMA_SetChannelPreemptionConfig (*C function*), 227
drivers_Kinetis_K_MK22F51212.EDMA_SetMajorOffsetConfig (*C function*), 229
drivers_Kinetis_K_MK22F51212.EDMA_SetMinorOffsetConfig (*C function*), 227
drivers_Kinetis_K_MK22F51212.EDMA_SetModulo (*C function*), 228
drivers_Kinetis_K_MK22F51212.EDMA_SetTransferConfig (*C function*), 227
drivers_Kinetis_K_MK22F51212.EDMA_StartTransfer (*C function*), 236
drivers_Kinetis_K_MK22F51212.EDMA_StopTransfer (*C function*), 236
drivers_Kinetis_K_MK22F51212.EDMA_SubmitTransfer (*C function*), 235
drivers_Kinetis_K_MK22F51212.edma_tcd_t (*C type*), 243
drivers_Kinetis_K_MK22F51212.EDMA_TcdDisableInterrupts (*C function*), 232
drivers_Kinetis_K_MK22F51212.EDMA_TcdEnableAutoStopRequest (*C function*), 232
drivers_Kinetis_K_MK22F51212.EDMA_TcdEnableInterrupts (*C function*), 232
drivers_Kinetis_K_MK22F51212.EDMA_TcdReset (*C function*), 230
drivers_Kinetis_K_MK22F51212.EDMA_TcdSetBandWidth (*C function*), 231
drivers_Kinetis_K_MK22F51212.EDMA_TcdSetChannelLink (*C function*), 231
drivers_Kinetis_K_MK22F51212.EDMA_TcdSetMajorOffsetConfig (*C function*), 232
drivers_Kinetis_K_MK22F51212.EDMA_TcdSetMinorOffsetConfig (*C function*), 230
drivers_Kinetis_K_MK22F51212.EDMA_TcdSetModulo (*C function*), 231
drivers_Kinetis_K_MK22F51212.EDMA_TcdSetTransferConfig (*C function*), 230
drivers_Kinetis_K_MK22F51212.edma_transfer_config_t (*C type*), 243

drivers_Kinetis_K_MK22F51212.edma_transfer_size_t (*C type*), 242
drivers_Kinetis_K_MK22F51212.edma_transfer_type_t (*C type*), 242
drivers_Kinetis_K_MK22F51212.EDMA_TriggerChannelStart (*C function*), 233
drivers_Kinetis_K_MK22F51212.EnableGlobalIRQ (*C function*), 366
drivers_Kinetis_K_MK22F51212.EnableIRQ (*C function*), 365
drivers_Kinetis_K_MK22F51212.EnableIRQWithPriority (*C function*), 365
drivers_Kinetis_K_MK22F51212.enableMode (*C var*), 174, 175
drivers_Kinetis_K_MK22F51212.er32kSrc (*C var*), 174
drivers_Kinetis_K_MK22F51212.erclkDiv (*C var*), 175
drivers_Kinetis_K_MK22F51212.EWM_CLOCKS (*C macro*), 162
drivers_Kinetis_K_MK22F51212.ewm_config_t (*C type*), 249
drivers_Kinetis_K_MK22F51212.EWM_Deinit (*C function*), 247
drivers_Kinetis_K_MK22F51212.EWM_DisableInterrupts (*C function*), 248
drivers_Kinetis_K_MK22F51212.EWM_EnableInterrupts (*C function*), 248
drivers_Kinetis_K_MK22F51212.EWM_GetDefaultConfig (*C function*), 247
drivers_Kinetis_K_MK22F51212.EWM_GetStatusFlags (*C function*), 248
drivers_Kinetis_K_MK22F51212.EWM_Init (*C function*), 247
drivers_Kinetis_K_MK22F51212.ewm_lpo_clock_source_t (*C type*), 249
drivers_Kinetis_K_MK22F51212.EWM_Refresh (*C function*), 249
drivers_Kinetis_K_MK22F51212.FAST_CLK (*C macro*), 164
drivers_Kinetis_K_MK22F51212.fcrdiv (*C var*), 175
drivers_Kinetis_K_MK22F51212.flash_config_t (*C type*), 291
drivers_Kinetis_K_MK22F51212.FLASH_Erase (*C function*), 278
drivers_Kinetis_K_MK22F51212.FLASH_EraseAll (*C function*), 279
drivers_Kinetis_K_MK22F51212.FLASH_EraseAllUnsecure (*C function*), 279
drivers_Kinetis_K_MK22F51212.FLASH_EraseSectorNonBlocking (*C function*), 278
drivers_Kinetis_K_MK22F51212.FLASH_GetCommandState (*C function*), 288
drivers_Kinetis_K_MK22F51212.FLASH_GetProperty (*C function*), 288
drivers_Kinetis_K_MK22F51212.FLASH_GetSecurityState (*C function*), 285
drivers_Kinetis_K_MK22F51212.FLASH_Init (*C function*), 277
drivers_Kinetis_K_MK22F51212.FLASH_IsExecuteOnly (*C function*), 287
drivers_Kinetis_K_MK22F51212.FLASH_IsProtected (*C function*), 286
drivers_Kinetis_K_MK22F51212.FLASH_PflashGetProtection (*C function*), 288
drivers_Kinetis_K_MK22F51212.FLASH_PflashSetProtection (*C function*), 287
drivers_Kinetis_K_MK22F51212.FLASH_Program (*C function*), 280
drivers_Kinetis_K_MK22F51212.FLASH_ProgramOnce (*C function*), 280
drivers_Kinetis_K_MK22F51212.FLASH_ProgramSection (*C function*), 281
drivers_Kinetis_K_MK22F51212.flash_property_tag_t (*C type*), 291
drivers_Kinetis_K_MK22F51212.flash_prot_state_t (*C type*), 290
drivers_Kinetis_K_MK22F51212.FLASH_ReadOnce (*C function*), 282
drivers_Kinetis_K_MK22F51212.FLASH_ReadResource (*C function*), 282
drivers_Kinetis_K_MK22F51212.FLASH_SecurityBypass (*C function*), 285
drivers_Kinetis_K_MK22F51212.FLASH_SetFlexramFunction (*C function*), 285
drivers_Kinetis_K_MK22F51212.FLASH_Swap (*C function*), 286
drivers_Kinetis_K_MK22F51212.FLASH_VerifyErase (*C function*), 283
drivers_Kinetis_K_MK22F51212.FLASH_VerifyEraseAll (*C function*), 284
drivers_Kinetis_K_MK22F51212.FLASH_VerifyProgram (*C function*), 284
drivers_Kinetis_K_MK22F51212.flash_xacc_state_t (*C type*), 291
drivers_Kinetis_K_MK22F51212.flexbus_address_setup_t (*C type*), 254
drivers_Kinetis_K_MK22F51212.flexbus_bytelane_shift_t (*C type*), 255
drivers_Kinetis_K_MK22F51212.FLEXBUS_CLOCKS (*C macro*), 162
drivers_Kinetis_K_MK22F51212.flexbus_config_t (*C type*), 255
drivers_Kinetis_K_MK22F51212.FLEXBUS_Deinit (*C function*), 251
drivers_Kinetis_K_MK22F51212.FLEXBUS_GetDefaultConfig (*C function*), 251
drivers_Kinetis_K_MK22F51212.FLEXBUS_Init (*C function*), 250
drivers_Kinetis_K_MK22F51212.flexbus_multiplex_group1_t (*C type*), 255
drivers_Kinetis_K_MK22F51212.flexbus_multiplex_group2_t (*C type*), 255
drivers_Kinetis_K_MK22F51212.flexbus_multiplex_group3_t (*C type*), 255

drivers_Kinetis_K_MK22F51212.flexbus_multiplex_group4_t (*C type*), 255
drivers_Kinetis_K_MK22F51212.flexbus_multiplex_group5_t (*C type*), 255
drivers_Kinetis_K_MK22F51212.flexbus_port_size_t (*C type*), 254
drivers_Kinetis_K_MK22F51212.flexbus_read_address_hold_t (*C type*), 254
drivers_Kinetis_K_MK22F51212.flexbus_write_address_hold_t (*C type*), 254
drivers_Kinetis_K_MK22F51212.flexnvm_config_t (*C type*), 302
drivers_Kinetis_K_MK22F51212.FLEXNVM_DflashErase (*C function*), 292
drivers_Kinetis_K_MK22F51212.FLEXNVM_DflashGetProtection (*C function*), 300
drivers_Kinetis_K_MK22F51212.FLEXNVM_DflashProgram (*C function*), 293
drivers_Kinetis_K_MK22F51212.FLEXNVM_DflashProgramSection (*C function*), 294
drivers_Kinetis_K_MK22F51212.FLEXNVM_DflashSetProtection (*C function*), 300
drivers_Kinetis_K_MK22F51212.FLEXNVM_DflashVerifyErase (*C function*), 297
drivers_Kinetis_K_MK22F51212.FLEXNVM_DflashVerifyProgram (*C function*), 298
drivers_Kinetis_K_MK22F51212.FLEXNVM_EepromGetProtection (*C function*), 301
drivers_Kinetis_K_MK22F51212.FLEXNVM_EepromSetProtection (*C function*), 300
drivers_Kinetis_K_MK22F51212.FLEXNVM_EepromWrite (*C function*), 302
drivers_Kinetis_K_MK22F51212.FLEXNVM_EraseAll (*C function*), 292
drivers_Kinetis_K_MK22F51212.FLEXNVM_EraseAllUnsecure (*C function*), 293
drivers_Kinetis_K_MK22F51212.FLEXNVM_GetProperty (*C function*), 301
drivers_Kinetis_K_MK22F51212.FLEXNVM_GetSecurityState (*C function*), 298
drivers_Kinetis_K_MK22F51212.FLEXNVM_Init (*C function*), 292
drivers_Kinetis_K_MK22F51212.FLEXNVM_ProgramPartition (*C function*), 295
drivers_Kinetis_K_MK22F51212.FLEXNVM_ProgramPartition_CSE (*C function*), 295
drivers_Kinetis_K_MK22F51212.flexnvm_property_tag_t (*C type*), 302
drivers_Kinetis_K_MK22F51212.FLEXNVM_ReadResource (*C function*), 296
drivers_Kinetis_K_MK22F51212.FLEXNVM_SecurityBypass (*C function*), 299
drivers_Kinetis_K_MK22F51212.FLEXNVM_SetFlexramFunction (*C function*), 299
drivers_Kinetis_K_MK22F51212.FLEXNVM_VerifyEraseAll (*C function*), 297
drivers_Kinetis_K_MK22F51212.FOUR_CHAR_CODE (*C macro*), 303
drivers_Kinetis_K_MK22F51212.frdiv (*C var*), 175
drivers_Kinetis_K_MK22F51212.freq (*C var*), 175
drivers_Kinetis_K_MK22F51212.FSL_ADC16_DRIVER_VERSION (*C macro*), 142
drivers_Kinetis_K_MK22F51212.FSL_CLOCK_DRIVER_VERSION (*C macro*), 162
drivers_Kinetis_K_MK22F51212.FSL_CMP_DRIVER_VERSION (*C macro*), 179
drivers_Kinetis_K_MK22F51212.FSL_COMMON_DRIVER_VERSION (*C macro*), 351
drivers_Kinetis_K_MK22F51212.FSL_COMPONENT_ID (*C macro*), 415, 447
drivers_Kinetis_K_MK22F51212.FSL_CRC_DRIVER_VERSION (*C macro*), 182
drivers_Kinetis_K_MK22F51212.FSL_DAC_DRIVER_VERSION (*C macro*), 188
drivers_Kinetis_K_MK22F51212.FSL_DMAMUX_DRIVER_VERSION (*C macro*), 191
drivers_Kinetis_K_MK22F51212.FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ (*C macro*), 367
drivers_Kinetis_K_MK22F51212.FSL_DSPI_DRIVER_VERSION (*C macro*), 204
drivers_Kinetis_K_MK22F51212.FSL_DSPI_EDMA_DRIVER_VERSION (*C macro*), 222
drivers_Kinetis_K_MK22F51212.FSL_EDMA_DRIVER_VERSION (*C macro*), 237
drivers_Kinetis_K_MK22F51212.FSL_EWM_DRIVER_VERSION (*C macro*), 249
drivers_Kinetis_K_MK22F51212.FSL_FLASH_DRIVER_VERSION (*C macro*), 289
drivers_Kinetis_K_MK22F51212.FSL_FLASH_DRIVER_VERSION_ROM (*C macro*), 289
drivers_Kinetis_K_MK22F51212.FSL_FLEXBUS_DRIVER_VERSION (*C macro*), 251
drivers_Kinetis_K_MK22F51212.FSL_FTM_DRIVER_VERSION (*C macro*), 312
drivers_Kinetis_K_MK22F51212.FSL_GPIO_DRIVER_VERSION (*C macro*), 330
drivers_Kinetis_K_MK22F51212.FSL_I2C_DRIVER_VERSION (*C macro*), 342
drivers_Kinetis_K_MK22F51212.FSL_I2C_EDMA_DRIVER_VERSION (*C macro*), 350
drivers_Kinetis_K_MK22F51212.FSL_LIN_LPUART_DRIVER_VERSION (*C macro*), 367
drivers_Kinetis_K_MK22F51212.FSL_LLWU_DRIVER_VERSION (*C macro*), 378
drivers_Kinetis_K_MK22F51212.FSL_LPTMR_DRIVER_VERSION (*C macro*), 383
drivers_Kinetis_K_MK22F51212.FSL_LPUART_DRIVER_VERSION (*C macro*), 398
drivers_Kinetis_K_MK22F51212.FSL_LPUART_EDMA_DRIVER_VERSION (*C macro*), 410
drivers_Kinetis_K_MK22F51212.FSL_MCM_DRIVER_VERSION (*C macro*), 411

drivers_Kinetis_K_MK22F51212.FSL_PDB_DRIVER_VERSION (*C macro*), 421
drivers_Kinetis_K_MK22F51212.FSL_PIT_DRIVER_VERSION (*C macro*), 431
drivers_Kinetis_K_MK22F51212.FSL_PMC_DRIVER_VERSION (*C macro*), 436
drivers_Kinetis_K_MK22F51212.FSL_PORT_DRIVER_VERSION (*C macro*), 443
drivers_Kinetis_K_MK22F51212.FSL_RCM_DRIVER_VERSION (*C macro*), 450
drivers_Kinetis_K_MK22F51212.FSL_RNGA_DRIVER_VERSION (*C macro*), 454
drivers_Kinetis_K_MK22F51212.FSL_RTC_DRIVER_VERSION (*C macro*), 459
drivers_Kinetis_K_MK22F51212.FSL_SAI_DRIVER_VERSION (*C macro*), 479
drivers_Kinetis_K_MK22F51212.FSL_SAI_EDMA_DRIVER_VERSION (*C macro*), 500
drivers_Kinetis_K_MK22F51212.FSL_SAI_HAS_FIFO_EXTEND_FEATURE (*C macro*), 489
drivers_Kinetis_K_MK22F51212.FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL (*C macro*), 176
drivers_Kinetis_K_MK22F51212.FSL_SIM_DRIVER_VERSION (*C macro*), 501
drivers_Kinetis_K_MK22F51212.FSL_SMARTCARD_DRIVER_VERSION (*C macro*), 503
drivers_Kinetis_K_MK22F51212.FSL_SMC_DRIVER_VERSION (*C macro*), 520
drivers_Kinetis_K_MK22F51212.FSL_UART_DRIVER_VERSION (*C macro*), 535
drivers_Kinetis_K_MK22F51212.FSL_UART_EDMA_DRIVER_VERSION (*C macro*), 544
drivers_Kinetis_K_MK22F51212.FSL_VREF_DRIVER_VERSION (*C macro*), 547
drivers_Kinetis_K_MK22F51212.FSL_WDOG_DRIVER_VERSION (*C macro*), 553
drivers_Kinetis_K_MK22F51212.FTF_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.FTFx_API_Init (*C function*), 260
drivers_Kinetis_K_MK22F51212.FTFx_API_UpdateFlexnvmPartitionStatus (*C function*), 261
drivers_Kinetis_K_MK22F51212.FTFx_CACHE_ClearCachePrefetchSpeculation (*C function*), 257
drivers_Kinetis_K_MK22F51212.ftfx_cache_config_t (*C type*), 257
drivers_Kinetis_K_MK22F51212.FTFx_CACHE_Init (*C function*), 257
drivers_Kinetis_K_MK22F51212.FTFx_CACHE_PflashGetPrefetchSpeculation (*C function*), 258
drivers_Kinetis_K_MK22F51212.FTFx_CACHE_PflashSetPrefetchSpeculation (*C function*), 257
drivers_Kinetis_K_MK22F51212.FTFx_CMD_Erase (*C function*), 261
drivers_Kinetis_K_MK22F51212.FTFx_CMD_EraseAll (*C function*), 262
drivers_Kinetis_K_MK22F51212.FTFx_CMD_EraseAllExecuteOnlySegments (*C function*), 263
drivers_Kinetis_K_MK22F51212.FTFx_CMD_EraseAllUnsecure (*C function*), 262
drivers_Kinetis_K_MK22F51212.FTFx_CMD_EraseSectorNonBlocking (*C function*), 261
drivers_Kinetis_K_MK22F51212.FTFx_CMD_Program (*C function*), 263
drivers_Kinetis_K_MK22F51212.FTFx_CMD_ProgramOnce (*C function*), 264
drivers_Kinetis_K_MK22F51212.FTFx_CMD_ProgramPartition (*C function*), 265
drivers_Kinetis_K_MK22F51212.FTFx_CMD_ProgramSection (*C function*), 264
drivers_Kinetis_K_MK22F51212.FTFx_CMD_ReadOnce (*C function*), 266
drivers_Kinetis_K_MK22F51212.FTFx_CMD_ReadResource (*C function*), 266
drivers_Kinetis_K_MK22F51212.FTFx_CMD_SecurityBypass (*C function*), 269
drivers_Kinetis_K_MK22F51212.FTFx_CMD_SetFlexramFunction (*C function*), 269
drivers_Kinetis_K_MK22F51212.FTFx_CMD_SwapControl (*C function*), 270
drivers_Kinetis_K_MK22F51212.FTFx_CMD_VerifyErase (*C function*), 267
drivers_Kinetis_K_MK22F51212.FTFx_CMD_VerifyEraseAll (*C function*), 267
drivers_Kinetis_K_MK22F51212.FTFx_CMD_VerifyEraseAllExecuteOnlySegments (*C function*), 268
drivers_Kinetis_K_MK22F51212.FTFx_CMD_VerifyProgram (*C function*), 268
drivers_Kinetis_K_MK22F51212.ftfx_config_t (*C type*), 274
drivers_Kinetis_K_MK22F51212.FTFx_DRIVER_HAS_FLASH1_SUPPORT (*C macro*), 277
drivers_Kinetis_K_MK22F51212.FTFx_DRIVER_IS_EXPORTED (*C macro*), 277
drivers_Kinetis_K_MK22F51212.FTFx_DRIVER_IS_FLASH_RESIDENT (*C macro*), 277
drivers_Kinetis_K_MK22F51212.FTFx_FLASH1_HAS_PROT_CONTROL (*C macro*), 277
drivers_Kinetis_K_MK22F51212.FTFx_FLASH1_HAS_XACC_CONTROL (*C macro*), 277
drivers_Kinetis_K_MK22F51212.FTFx_FLASH1_IS_INDEPENDENT_BLOCK (*C macro*), 277
drivers_Kinetis_K_MK22F51212.FTFx_FLASH_COUNT (*C macro*), 277
drivers_Kinetis_K_MK22F51212.ftfx_flexram_func_opt_t (*C type*), 273
drivers_Kinetis_K_MK22F51212.ftfx_ifr_desc_t (*C type*), 274
drivers_Kinetis_K_MK22F51212.ftfx_margin_value_t (*C type*), 273
drivers_Kinetis_K_MK22F51212.ftfx_mem_desc_t (*C type*), 274

drivers_Kinetis_K_MK22F51212.ftfx_ops_config_t (*C type*), 274
drivers_Kinetis_K_MK22F51212.ftfx_partition_flexram_load_opt_t (*C type*), 273
drivers_Kinetis_K_MK22F51212.ftfx_prefetch_speculation_status_t (*C type*), 257
drivers_Kinetis_K_MK22F51212.ftfx_read_resource_opt_t (*C type*), 273
drivers_Kinetis_K_MK22F51212.FTFx_REG_GetSecurityState (*C function*), 269
drivers_Kinetis_K_MK22F51212.ftfx_security_state_t (*C type*), 273
drivers_Kinetis_K_MK22F51212.ftfx_spec_mem_t (*C type*), 274
drivers_Kinetis_K_MK22F51212.ftfx_swap_block_status_t (*C type*), 274
drivers_Kinetis_K_MK22F51212.ftfx_swap_control_opt_t (*C type*), 274
drivers_Kinetis_K_MK22F51212.ftfx_swap_state_config_t (*C type*), 274
drivers_Kinetis_K_MK22F51212.ftfx_swap_state_t (*C type*), 274
drivers_Kinetis_K_MK22F51212.ftm_bdm_mode_t (*C type*), 323
drivers_Kinetis_K_MK22F51212.FTM_CalculateCounterClkDiv (*C function*), 304
drivers_Kinetis_K_MK22F51212.ftm_channel_index_t (*C type*), 323
drivers_Kinetis_K_MK22F51212.ftm_chnl_param_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.ftm_chnl_pwm_config_param_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.ftm_chnl_pwm_signal_param_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.ftm_chnl_t (*C type*), 321
drivers_Kinetis_K_MK22F51212.FTM_ClearQuadDecoderCounterValue (*C function*), 312
drivers_Kinetis_K_MK22F51212.FTM_ClearStatusFlags (*C function*), 308
drivers_Kinetis_K_MK22F51212.ftm_clock_prescale_t (*C type*), 323
drivers_Kinetis_K_MK22F51212.ftm_clock_source_t (*C type*), 323
drivers_Kinetis_K_MK22F51212.FTM_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.ftm_config_t (*C type*), 323
drivers_Kinetis_K_MK22F51212.FTM_ConfigCombinePWM (*C function*), 306
drivers_Kinetis_K_MK22F51212.FTM_ConfigSinglePWM (*C function*), 306
drivers_Kinetis_K_MK22F51212.ftm_deadtime_prescale_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.FTM_Deinit (*C function*), 304
drivers_Kinetis_K_MK22F51212.FTM_DisableInterrupts (*C function*), 308
drivers_Kinetis_K_MK22F51212.ftm_dual_edge_capture_mode_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.ftm_dual_edge_capture_param_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.FTM_EnableDmaTransfer (*C function*), 325
drivers_Kinetis_K_MK22F51212.FTM_EnableInterrupts (*C function*), 307
drivers_Kinetis_K_MK22F51212.ftm_external_trigger_t (*C type*), 323
drivers_Kinetis_K_MK22F51212.ftm_fault_input_t (*C type*), 321
drivers_Kinetis_K_MK22F51212.ftm_fault_mode_t (*C type*), 323
drivers_Kinetis_K_MK22F51212.ftm_fault_param_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.FTM_GetCurrentTimerCount (*C function*), 309
drivers_Kinetis_K_MK22F51212.FTM_GetDefaultConfig (*C function*), 304
drivers_Kinetis_K_MK22F51212.FTM_GetEnabledInterrupts (*C function*), 308
drivers_Kinetis_K_MK22F51212.FTM_GetInputCaptureValue (*C function*), 309
drivers_Kinetis_K_MK22F51212.FTM_GetInstance (*C function*), 308
drivers_Kinetis_K_MK22F51212.FTM_GetQuadDecoderCounterValue (*C function*), 312
drivers_Kinetis_K_MK22F51212.FTM_GetSoftwareOutputEnable (*C function*), 310
drivers_Kinetis_K_MK22F51212.FTM_GetSoftwareOutputValue (*C function*), 310
drivers_Kinetis_K_MK22F51212.FTM_GetStatusFlags (*C function*), 308
drivers_Kinetis_K_MK22F51212.FTM_Init (*C function*), 304
drivers_Kinetis_K_MK22F51212.ftm_input_capture_edge_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.ftm_interrupt_enable_t (*C type*), 323
drivers_Kinetis_K_MK22F51212.ftm_output_compare_mode_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.ftm_phase_params_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.ftm_phase_polarity_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.ftm_pwm_level_select_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.ftm_pwm_mode_t (*C type*), 321
drivers_Kinetis_K_MK22F51212.ftm_pwm_sync_method_t (*C type*), 323
drivers_Kinetis_K_MK22F51212.ftm_quad_decode_mode_t (*C type*), 322
drivers_Kinetis_K_MK22F51212.ftm_reload_point_t (*C type*), 323
drivers_Kinetis_K_MK22F51212.FTM_SetChannelMatchValue (*C function*), 309

drivers_Kinetis_K_MK22F51212.FTM_SetComplementaryEnable (*C function*), 311
drivers_Kinetis_K_MK22F51212.FTM_SetDeadTimeEnable (*C function*), 311
drivers_Kinetis_K_MK22F51212.FTM_SetFaultControlEnable (*C function*), 311
drivers_Kinetis_K_MK22F51212.FTM_SetGlobalTimeBaseOutputEnable (*C function*), 324
drivers_Kinetis_K_MK22F51212.FTM_SetHalfCycReloadMatchValue (*C function*), 325
drivers_Kinetis_K_MK22F51212.FTM_SetInitialModuloValue (*C function*), 309
drivers_Kinetis_K_MK22F51212.FTM_SetInvertEnable (*C function*), 311
drivers_Kinetis_K_MK22F51212.FTM_SetLdok (*C function*), 325
drivers_Kinetis_K_MK22F51212.FTM_SetLoadFreq (*C function*), 325
drivers_Kinetis_K_MK22F51212.FTM_SetOutputMask (*C function*), 324
drivers_Kinetis_K_MK22F51212.FTM_SetPwmOutputEnable (*C function*), 324
drivers_Kinetis_K_MK22F51212.FTM_SetQuadDecoderModuloValue (*C function*), 312
drivers_Kinetis_K_MK22F51212.FTM_SetSoftwareCtrlEnable (*C function*), 310
drivers_Kinetis_K_MK22F51212.FTM_SetSoftwareCtrlVal (*C function*), 310
drivers_Kinetis_K_MK22F51212.FTM_SetSoftwareOutputCtrl (*C function*), 310
drivers_Kinetis_K_MK22F51212.FTM_SetSoftwareTrigger (*C function*), 324
drivers_Kinetis_K_MK22F51212.FTM_SetTimerPeriod (*C function*), 308
drivers_Kinetis_K_MK22F51212.FTM_SetupDualEdgeCapture (*C function*), 307
drivers_Kinetis_K_MK22F51212.FTM_SetupFaultInput (*C function*), 324
drivers_Kinetis_K_MK22F51212.FTM_SetupInputCapture (*C function*), 307
drivers_Kinetis_K_MK22F51212.FTM_SetupOutputCompare (*C function*), 307
drivers_Kinetis_K_MK22F51212.FTM_SetupPwm (*C function*), 304
drivers_Kinetis_K_MK22F51212.FTM_SetupPwmMode (*C function*), 306
drivers_Kinetis_K_MK22F51212.FTM_SetupQuadDecode (*C function*), 311
drivers_Kinetis_K_MK22F51212.FTM_SetWriteProtection (*C function*), 324
drivers_Kinetis_K_MK22F51212.FTM_StartTimer (*C function*), 309
drivers_Kinetis_K_MK22F51212.ftm_status_flags_t (*C type*), 323
drivers_Kinetis_K_MK22F51212.FTM_StopTimer (*C function*), 310
drivers_Kinetis_K_MK22F51212.FTM_UpdateChnlEdgeLevelSelect (*C function*), 306
drivers_Kinetis_K_MK22F51212.FTM_UpdatePwmDutycycle (*C function*), 305
drivers_Kinetis_K_MK22F51212.function_bit_operation_ptr_t (*C union*), 258
drivers_Kinetis_K_MK22F51212.function_bit_operation_ptr_t.callFlashCommand (*C var*), 258
drivers_Kinetis_K_MK22F51212.function_bit_operation_ptr_t.commadAddr (*C var*), 258
drivers_Kinetis_K_MK22F51212.function_ptr_t (*C union*), 275
drivers_Kinetis_K_MK22F51212.function_ptr_t.callFlashCommand (*C var*), 276
drivers_Kinetis_K_MK22F51212.function_ptr_t.commadAddr (*C var*), 276
drivers_Kinetis_K_MK22F51212.g_dspiDummyData (*C var*), 213
drivers_Kinetis_K_MK22F51212.g_xtal0Freq (*C var*), 159
drivers_Kinetis_K_MK22F51212.g_xtal32Freq (*C var*), 159
drivers_Kinetis_K_MK22F51212.GPIO_CheckAttributeBytes (*C function*), 333
drivers_Kinetis_K_MK22F51212.gpio_checker_attribute_t (*C type*), 330
drivers_Kinetis_K_MK22F51212.GPIO_FIT_REG (*C macro*), 331
drivers_Kinetis_K_MK22F51212.gpio_pin_config_t (*C type*), 331
drivers_Kinetis_K_MK22F51212.gpio_pin_direction_t (*C type*), 330
drivers_Kinetis_K_MK22F51212.GPIO_PinInit (*C function*), 331
drivers_Kinetis_K_MK22F51212.GPIO_PinRead (*C function*), 332
drivers_Kinetis_K_MK22F51212.GPIO_PinWrite (*C function*), 332
drivers_Kinetis_K_MK22F51212.GPIO_PortClear (*C function*), 332
drivers_Kinetis_K_MK22F51212.GPIO_PortClearInterruptFlags (*C function*), 333
drivers_Kinetis_K_MK22F51212.GPIO_PortDenit (*C function*), 331
drivers_Kinetis_K_MK22F51212.GPIO_PortGetInterruptFlags (*C function*), 333
drivers_Kinetis_K_MK22F51212.GPIO_PortInit (*C function*), 331
drivers_Kinetis_K_MK22F51212.GPIO_PortSet (*C function*), 332
drivers_Kinetis_K_MK22F51212.GPIO_PortToggle (*C function*), 332
drivers_Kinetis_K_MK22F51212.I2C0_CLK_SRC (*C macro*), 164
drivers_Kinetis_K_MK22F51212.I2C1_CLK_SRC (*C macro*), 164
drivers_Kinetis_K_MK22F51212.I2C_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.i2c_direction_t (*C type*), 345

drivers_Kinetis_K_MK22F51212.I2C_DisableInterrupts (*C function*), 336
drivers_Kinetis_K_MK22F51212.I2C_DriverIRQHandler (*C function*), 342
drivers_Kinetis_K_MK22F51212.I2C_Enable (*C function*), 335
drivers_Kinetis_K_MK22F51212.I2C_EnableDMA (*C function*), 337
drivers_Kinetis_K_MK22F51212.I2C_EnableInterrupts (*C function*), 336
drivers_Kinetis_K_MK22F51212.I2C_GetDataRegAddr (*C function*), 337
drivers_Kinetis_K_MK22F51212.I2C_GetInstance (*C function*), 335
drivers_Kinetis_K_MK22F51212.I2C_HAS_STOP_DETECT (*C macro*), 346
drivers_Kinetis_K_MK22F51212.i2c_master_config_t (*C type*), 346
drivers_Kinetis_K_MK22F51212.i2c_master_edma_handle_t (*C type*), 350
drivers_Kinetis_K_MK22F51212.i2c_master_edma_transfer_callback_t (*C type*), 350
drivers_Kinetis_K_MK22F51212.I2C_MASTER_FACK_CONTROL (*C macro*), 346
drivers_Kinetis_K_MK22F51212.i2c_master_handle_t (*C type*), 346
drivers_Kinetis_K_MK22F51212.i2c_master_transfer_callback_t (*C type*), 346
drivers_Kinetis_K_MK22F51212.i2c_master_transfer_t (*C type*), 346
drivers_Kinetis_K_MK22F51212.I2C_MasterClearStatusFlags (*C function*), 335
drivers_Kinetis_K_MK22F51212.I2C_MasterCreateEDMAHandle (*C function*), 349
drivers_Kinetis_K_MK22F51212.I2C_MasterDeinit (*C function*), 334
drivers_Kinetis_K_MK22F51212.I2C_MasterGetDefaultConfig (*C function*), 335
drivers_Kinetis_K_MK22F51212.I2C_MasterGetStatusFlags (*C function*), 335
drivers_Kinetis_K_MK22F51212.I2C_MasterInit (*C function*), 333
drivers_Kinetis_K_MK22F51212.I2C_MasterReadBlocking (*C function*), 338
drivers_Kinetis_K_MK22F51212.I2C_MasterRepeatedStart (*C function*), 337
drivers_Kinetis_K_MK22F51212.I2C_MasterSetBaudRate (*C function*), 337
drivers_Kinetis_K_MK22F51212.I2C_MasterStart (*C function*), 337
drivers_Kinetis_K_MK22F51212.I2C_MasterStop (*C function*), 337
drivers_Kinetis_K_MK22F51212.I2C_MasterTransferAbort (*C function*), 340
drivers_Kinetis_K_MK22F51212.I2C_MasterTransferAbortEDMA (*C function*), 350
drivers_Kinetis_K_MK22F51212.I2C_MasterTransferBlocking (*C function*), 339
drivers_Kinetis_K_MK22F51212.I2C_MasterTransferCreateHandle (*C function*), 339
drivers_Kinetis_K_MK22F51212.I2C_MasterTransferEDMA (*C function*), 350
drivers_Kinetis_K_MK22F51212.I2C_MasterTransferGetCount (*C function*), 340
drivers_Kinetis_K_MK22F51212.I2C_MasterTransferGetCountEDMA (*C function*), 350
drivers_Kinetis_K_MK22F51212.I2C_MasterTransferHandleIRQ (*C function*), 340
drivers_Kinetis_K_MK22F51212.I2C_MasterTransferNonBlocking (*C function*), 339
drivers_Kinetis_K_MK22F51212.I2C_MasterWriteBlocking (*C function*), 338
drivers_Kinetis_K_MK22F51212.I2C_RETRY_TIMES (*C macro*), 346
drivers_Kinetis_K_MK22F51212.i2c_slave_address_mode_t (*C type*), 345
drivers_Kinetis_K_MK22F51212.i2c_slave_config_t (*C type*), 346
drivers_Kinetis_K_MK22F51212.i2c_slave_handle_t (*C type*), 346
drivers_Kinetis_K_MK22F51212.i2c_slave_transfer_callback_t (*C type*), 346
drivers_Kinetis_K_MK22F51212.i2c_slave_transfer_event_t (*C type*), 345
drivers_Kinetis_K_MK22F51212.i2c_slave_transfer_t (*C type*), 346
drivers_Kinetis_K_MK22F51212.I2C_SlaveClearStatusFlags (*C function*), 336
drivers_Kinetis_K_MK22F51212.I2C_SlaveDeinit (*C function*), 334
drivers_Kinetis_K_MK22F51212.I2C_SlaveGetDefaultConfig (*C function*), 335
drivers_Kinetis_K_MK22F51212.I2C_SlaveGetStatusFlags (*C function*), 335
drivers_Kinetis_K_MK22F51212.I2C_SlaveInit (*C function*), 334
drivers_Kinetis_K_MK22F51212.I2C_SlaveReadBlocking (*C function*), 339
drivers_Kinetis_K_MK22F51212.I2C_SlaveTransferAbort (*C function*), 341
drivers_Kinetis_K_MK22F51212.I2C_SlaveTransferCreateHandle (*C function*), 341
drivers_Kinetis_K_MK22F51212.I2C_SlaveTransferGetCount (*C function*), 341
drivers_Kinetis_K_MK22F51212.I2C_SlaveTransferHandleIRQ (*C function*), 342
drivers_Kinetis_K_MK22F51212.I2C_SlaveTransferNonBlocking (*C function*), 341
drivers_Kinetis_K_MK22F51212.I2C_SlaveWriteBlocking (*C function*), 338
drivers_Kinetis_K_MK22F51212.ircIkEnableMode (*C var*), 175
drivers_Kinetis_K_MK22F51212.ircs (*C var*), 175
drivers_Kinetis_K_MK22F51212.IRQ_ClearPendingIRQ (*C function*), 366

drivers_Kinetis_K_MK22F51212.IRQ_SetPriority (*C function*), 366
drivers_Kinetis_K_MK22F51212.kFLASH_ApiEraseKey (*C macro*), 291
drivers_Kinetis_K_MK22F51212.kStatus_FLASH_Success (*C macro*), 291
drivers_Kinetis_K_MK22F51212.LIN_LPUART_AbortTransferData (*C function*), 373
drivers_Kinetis_K_MK22F51212.LIN_LPUART_AutoBaudCapture (*C function*), 375
drivers_Kinetis_K_MK22F51212.lin_lpuart_bit_count_per_char_t (*C enum*), 371
drivers_Kinetis_K_MK22F51212.lin_lpuart_bit_count_per_char_t.LPUART_8_BITS_PER_CHAR (*C enumerator*), 371
drivers_Kinetis_K_MK22F51212.lin_lpuart_bit_count_per_char_t.LPUART_9_BITS_PER_CHAR (*C enumerator*), 371
drivers_Kinetis_K_MK22F51212.lin_lpuart_bit_count_per_char_t.LPUART_10_BITS_PER_CHAR (*C enumerator*), 371
drivers_Kinetis_K_MK22F51212.LIN_LPUART_CalculateBaudRate (*C function*), 371
drivers_Kinetis_K_MK22F51212.LIN_LPUART_Deinit (*C function*), 372
drivers_Kinetis_K_MK22F51212.LIN_LPUART_DisableIRQ (*C function*), 375
drivers_Kinetis_K_MK22F51212.LIN_LPUART_EnableIRQ (*C function*), 374
drivers_Kinetis_K_MK22F51212.LIN_LPUART_GetReceiveStatus (*C function*), 373
drivers_Kinetis_K_MK22F51212.LIN_LPUART_GetRxDataPolarity (*C function*), 371
drivers_Kinetis_K_MK22F51212.LIN_LPUART_GetTransmitStatus (*C function*), 373
drivers_Kinetis_K_MK22F51212.LIN_LPUART_GotoIdleState (*C function*), 374
drivers_Kinetis_K_MK22F51212.LIN_LPUART_GoToSleepMode (*C function*), 374
drivers_Kinetis_K_MK22F51212.LIN_LPUART_Init (*C function*), 371
drivers_Kinetis_K_MK22F51212.LIN_LPUART_IRQHandler (*C function*), 375
drivers_Kinetis_K_MK22F51212.LIN_LPUART_MasterSendHeader (*C function*), 374
drivers_Kinetis_K_MK22F51212.LIN_LPUART_ReadByte (*C function*), 371
drivers_Kinetis_K_MK22F51212.LIN_LPUART_RecvFrmData (*C function*), 373
drivers_Kinetis_K_MK22F51212.LIN_LPUART_RecvFrmDataBlocking (*C function*), 373
drivers_Kinetis_K_MK22F51212.LIN_LPUART_SendFrameData (*C function*), 372
drivers_Kinetis_K_MK22F51212.LIN_LPUART_SendFrameDataBlocking (*C function*), 372
drivers_Kinetis_K_MK22F51212.LIN_LPUART_SendWakeupSignal (*C function*), 374
drivers_Kinetis_K_MK22F51212.LIN_LPUART_SetBaudRate (*C function*), 371
drivers_Kinetis_K_MK22F51212.LIN_LPUART_SetRxDataPolarity (*C function*), 371
drivers_Kinetis_K_MK22F51212.lin_lpuart_stop_bit_count_t (*C type*), 371
drivers_Kinetis_K_MK22F51212.LIN_LPUART_TRANSMISSION_COMPLETE_TIMEOUT (*C macro*), 375
drivers_Kinetis_K_MK22F51212.LIN_LPUART_WriteByte (*C function*), 371
drivers_Kinetis_K_MK22F51212.LLWU_ClearExternalWakeupPinFlag (*C function*), 377
drivers_Kinetis_K_MK22F51212.LLWU_ClearPinFilterFlag (*C function*), 378
drivers_Kinetis_K_MK22F51212.LLWU_EnableInternalModuleDmaRequestWakup (*C function*), 377
drivers_Kinetis_K_MK22F51212.LLWU_EnableInternalModuleInterruptWakup (*C function*), 377
drivers_Kinetis_K_MK22F51212.llwu_external_pin_filter_mode_t (*C type*), 379
drivers_Kinetis_K_MK22F51212.llwu_external_pin_mode_t (*C type*), 379
drivers_Kinetis_K_MK22F51212.LLWU_GetExternalWakeupPinFlag (*C function*), 377
drivers_Kinetis_K_MK22F51212.LLWU_GetParam (*C function*), 376
drivers_Kinetis_K_MK22F51212.LLWU_GetPinFilterFlag (*C function*), 378
drivers_Kinetis_K_MK22F51212.LLWU_GetVersionId (*C function*), 376
drivers_Kinetis_K_MK22F51212.llwu_param_t (*C type*), 379
drivers_Kinetis_K_MK22F51212.llwu_pin_filter_mode_t (*C type*), 379
drivers_Kinetis_K_MK22F51212.LLWU_REG_VAL (*C macro*), 379
drivers_Kinetis_K_MK22F51212.LLWU_SetExternalWakeupPinMode (*C function*), 377
drivers_Kinetis_K_MK22F51212.LLWU_SetPinFilterMode (*C function*), 378
drivers_Kinetis_K_MK22F51212.LLWU_SetResetPinMode (*C function*), 378
drivers_Kinetis_K_MK22F51212.llwu_version_id_t (*C type*), 379
drivers_Kinetis_K_MK22F51212.LPO_CLK_FREQ (*C macro*), 164
drivers_Kinetis_K_MK22F51212.LPTMR_ClearStatusFlags (*C function*), 382
drivers_Kinetis_K_MK22F51212.LPTMR_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.lptmr_config_t (*C type*), 386
drivers_Kinetis_K_MK22F51212.LPTMR_Deinit (*C function*), 381

drivers_Kinetis_K_MK22F51212.LPTMR_DisableInterrupts (*C function*), 381
drivers_Kinetis_K_MK22F51212.LPTMR_EnableInterrupts (*C function*), 381
drivers_Kinetis_K_MK22F51212.LPTMR_EnableTimerDMA (*C function*), 386
drivers_Kinetis_K_MK22F51212.LPTMR_GetCurrentTimerCount (*C function*), 382
drivers_Kinetis_K_MK22F51212.LPTMR_GetDefaultConfig (*C function*), 381
drivers_Kinetis_K_MK22F51212.LPTMR_GetEnabledInterrupts (*C function*), 381
drivers_Kinetis_K_MK22F51212.LPTMR_GetStatusFlags (*C function*), 381
drivers_Kinetis_K_MK22F51212.LPTMR_Init (*C function*), 380
drivers_Kinetis_K_MK22F51212.lptmr_interrupt_enable_t (*C type*), 385
drivers_Kinetis_K_MK22F51212.lptmr_pin_polarity_t (*C type*), 385
drivers_Kinetis_K_MK22F51212.lptmr_pin_select_t (*C type*), 385
drivers_Kinetis_K_MK22F51212.lptmr_prescaler_clock_select_t (*C type*), 385
drivers_Kinetis_K_MK22F51212.lptmr_prescaler_glitch_value_t (*C type*), 385
drivers_Kinetis_K_MK22F51212.LPTMR_SetTimerPeriod (*C function*), 382
drivers_Kinetis_K_MK22F51212.LPTMR_StartTimer (*C function*), 382
drivers_Kinetis_K_MK22F51212.lptmr_status_flags_t (*C type*), 386
drivers_Kinetis_K_MK22F51212.LPTMR_StopTimer (*C function*), 382
drivers_Kinetis_K_MK22F51212.lptmr_timer_mode_t (*C type*), 385
drivers_Kinetis_K_MK22F51212.LPUART_ClearStatusFlags (*C function*), 390
drivers_Kinetis_K_MK22F51212.LPUART_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.lpuart_config_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.lpuart_data_bits_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.LPUART_Deinit (*C function*), 388
drivers_Kinetis_K_MK22F51212.LPUART_DisableInterrupts (*C function*), 391
drivers_Kinetis_K_MK22F51212.LPUART_DriverIRQHandler (*C function*), 398
drivers_Kinetis_K_MK22F51212.lpuart_edma_handle_t (*C type*), 410
drivers_Kinetis_K_MK22F51212.lpuart_edma_transfer_callback_t (*C type*), 410
drivers_Kinetis_K_MK22F51212.LPUART_Enable9bitMode (*C function*), 388
drivers_Kinetis_K_MK22F51212.LPUART_EnableInterrupts (*C function*), 390
drivers_Kinetis_K_MK22F51212.LPUART_EnableMatchAddress (*C function*), 389
drivers_Kinetis_K_MK22F51212.LPUART_EnableRx (*C function*), 392
drivers_Kinetis_K_MK22F51212.LPUART_EnableRxDMA (*C function*), 392
drivers_Kinetis_K_MK22F51212.LPUART_EnableTx (*C function*), 392
drivers_Kinetis_K_MK22F51212.LPUART_EnableTxDMA (*C function*), 391
drivers_Kinetis_K_MK22F51212.LPUART_GetDataRegisterAddress (*C function*), 391
drivers_Kinetis_K_MK22F51212.LPUART_GetDefaultConfig (*C function*), 388
drivers_Kinetis_K_MK22F51212.LPUART_GetEnabledInterrupts (*C function*), 391
drivers_Kinetis_K_MK22F51212.LPUART_GetInstance (*C function*), 392
drivers_Kinetis_K_MK22F51212.LPUART_GetRxFifoCount (*C function*), 393
drivers_Kinetis_K_MK22F51212.LPUART_GetStatusFlags (*C function*), 390
drivers_Kinetis_K_MK22F51212.LPUART_GetTxFifoCount (*C function*), 393
drivers_Kinetis_K_MK22F51212.lpuart_handle_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.lpuart_idle_config_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.lpuart_idle_type_select_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.LPUART_Init (*C function*), 387
drivers_Kinetis_K_MK22F51212.lpuart_isr_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.lpuart_parity_mode_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.LPUART_ReadBlocking (*C function*), 394
drivers_Kinetis_K_MK22F51212.LPUART_ReadBlocking16bit (*C function*), 394
drivers_Kinetis_K_MK22F51212.LPUART_ReadByte (*C function*), 392
drivers_Kinetis_K_MK22F51212.LPUART_ReceiveEDMA (*C function*), 409
drivers_Kinetis_K_MK22F51212.LPUART_SendAddress (*C function*), 393
drivers_Kinetis_K_MK22F51212.LPUART_SendEDMA (*C function*), 409
drivers_Kinetis_K_MK22F51212.LPUART_SetBaudRate (*C function*), 388
drivers_Kinetis_K_MK22F51212.LPUART_SetMatchAddress (*C function*), 389
drivers_Kinetis_K_MK22F51212.LPUART_SetRxFifoWatermark (*C function*), 389
drivers_Kinetis_K_MK22F51212.LPUART_SetTxFifoWatermark (*C function*), 389
drivers_Kinetis_K_MK22F51212.LPUART_SoftwareReset (*C function*), 387

drivers_Kinetis_K_MK22F51212.lpuart_stop_bit_count_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.lpuart_transfer_callback_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.lpuart_transfer_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.LPUART_TransferAbortReceive (*C function*), 397
drivers_Kinetis_K_MK22F51212.LPUART_TransferAbortReceiveEDMA (*C function*), 409
drivers_Kinetis_K_MK22F51212.LPUART_TransferAbortSend (*C function*), 396
drivers_Kinetis_K_MK22F51212.LPUART_TransferAbortSendEDMA (*C function*), 409
drivers_Kinetis_K_MK22F51212.LPUART_TransferCreateHandle (*C function*), 394
drivers_Kinetis_K_MK22F51212.LPUART_TransferCreateHandleEDMA (*C function*), 408
drivers_Kinetis_K_MK22F51212.LPUART_TransferEdmaHandleIRQ (*C function*), 410
drivers_Kinetis_K_MK22F51212.LPUART_TransferEnable16Bit (*C function*), 389
drivers_Kinetis_K_MK22F51212.LPUART_TransferGetReceiveCount (*C function*), 397
drivers_Kinetis_K_MK22F51212.LPUART_TransferGetReceiveCountEDMA (*C function*), 410
drivers_Kinetis_K_MK22F51212.LPUART_TransferGetRxRingBufferLength (*C function*), 396
drivers_Kinetis_K_MK22F51212.LPUART_TransferGetSendCount (*C function*), 396
drivers_Kinetis_K_MK22F51212.LPUART_TransferGetSendCountEDMA (*C function*), 409
drivers_Kinetis_K_MK22F51212.LPUART_TransferHandleErrorIRQ (*C function*), 398
drivers_Kinetis_K_MK22F51212.LPUART_TransferHandleIRQ (*C function*), 398
drivers_Kinetis_K_MK22F51212.LPUART_TransferReceiveNonBlocking (*C function*), 397
drivers_Kinetis_K_MK22F51212.LPUART_TransferSendNonBlocking (*C function*), 395
drivers_Kinetis_K_MK22F51212.LPUART_TransferStartRingBuffer (*C function*), 395
drivers_Kinetis_K_MK22F51212.LPUART_TransferStopRingBuffer (*C function*), 396
drivers_Kinetis_K_MK22F51212.lpuart_transmit_cts_config_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.lpuart_transmit_cts_source_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.lpuart_transmit_rts_polarity_t (*C type*), 404
drivers_Kinetis_K_MK22F51212.LPUART_WriteBlocking (*C function*), 393
drivers_Kinetis_K_MK22F51212.LPUART_WriteBlocking16bit (*C function*), 393
drivers_Kinetis_K_MK22F51212.LPUART_WriteByte (*C function*), 392
drivers_Kinetis_K_MK22F51212.MAKE_STATUS (*C macro*), 303, 367
drivers_Kinetis_K_MK22F51212.MAKE_VERSION (*C macro*), 303, 367
drivers_Kinetis_K_MK22F51212.MAX (*C macro*), 352
drivers_Kinetis_K_MK22F51212.mcg_atm_select_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.mcg_clkout_src_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.MCG_CONFIG_CHECK_PARAM (*C macro*), 176
drivers_Kinetis_K_MK22F51212.mcg_config_t (*C type*), 159
drivers_Kinetis_K_MK22F51212.mcg_dmx32_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.mcg_drs_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.mcg_fll_src_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.MCG_INTERNAL_IRC_48M (*C macro*), 162
drivers_Kinetis_K_MK22F51212.mcg_irc_mode_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.mcg_mode_t (*C type*), 159
drivers_Kinetis_K_MK22F51212.mcg_monitor_mode_t (*C type*), 159
drivers_Kinetis_K_MK22F51212.mcg_oscsel_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.mcg_pll_clk_select_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.mcg_pll_config_t (*C type*), 159
drivers_Kinetis_K_MK22F51212.mcg_pll_ref_src_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.mcgMode (*C var*), 175
drivers_Kinetis_K_MK22F51212.mcm_buffer_fault_attribute_t (*C type*), 412
drivers_Kinetis_K_MK22F51212.MCM_BypassFixedCodeCacheMap (*C function*), 413
drivers_Kinetis_K_MK22F51212.MCM_ClearCacheWriteBufferErroStatus (*C function*), 413
drivers_Kinetis_K_MK22F51212.MCM_ClearCodeBusCache (*C function*), 414
drivers_Kinetis_K_MK22F51212.MCM_DisableInterruptStatus (*C function*), 412
drivers_Kinetis_K_MK22F51212.MCM_EnableCacheParityReporting (*C function*), 414
drivers_Kinetis_K_MK22F51212.MCM_EnableCodeBusCache (*C function*), 413
drivers_Kinetis_K_MK22F51212.MCM_EnableCodeCacheWriteBuffer (*C function*), 414
drivers_Kinetis_K_MK22F51212.MCM_EnableCrossbarRoundRobin (*C function*), 412
drivers_Kinetis_K_MK22F51212.MCM_EnableInterruptStatus (*C function*), 412
drivers_Kinetis_K_MK22F51212.MCM_EnablePcParity (*C function*), 414

drivers_Kinetis_K_MK22F51212.MCM_EnablePcParityFaultReport (*C function*), 414
drivers_Kinetis_K_MK22F51212.MCM_ForceCodeCacheToNoAllocation (*C function*), 414
drivers_Kinetis_K_MK22F51212.MCM_GetBufferFaultAddress (*C function*), 413
drivers_Kinetis_K_MK22F51212.MCM_GetBufferFaultAttribute (*C function*), 413
drivers_Kinetis_K_MK22F51212.MCM_GetBufferFaultData (*C function*), 413
drivers_Kinetis_K_MK22F51212.MCM_GetInterruptStatus (*C function*), 413
drivers_Kinetis_K_MK22F51212.MCM_GetLmemFaultAddress (*C function*), 415
drivers_Kinetis_K_MK22F51212.MCM_GetLmemFaultAttribute (*C function*), 415
drivers_Kinetis_K_MK22F51212.MCM_GetLmemFaultData (*C function*), 415
drivers_Kinetis_K_MK22F51212.MCM_ISCR_STAT_MASK (*C macro*), 415
drivers_Kinetis_K_MK22F51212.MCM_LimitCodeCachePeripheralWriteBuffering (*C function*), 413
drivers_Kinetis_K_MK22F51212.mcm_lmem_fault_attribute_t (*C type*), 412
drivers_Kinetis_K_MK22F51212.MCM_LMFATR_BUFF_MASK (*C macro*), 415
drivers_Kinetis_K_MK22F51212.MCM_LMFATR_CACH_MASK (*C macro*), 415
drivers_Kinetis_K_MK22F51212.MCM_LMFATR_MODE_MASK (*C macro*), 415
drivers_Kinetis_K_MK22F51212.MCM_LMFATR_TYPE_MASK (*C macro*), 415
drivers_Kinetis_K_MK22F51212.MCM_LockConfigState (*C function*), 414
drivers_Kinetis_K_MK22F51212.MCUX_DEPRECATED (*C macro*), 355
drivers_Kinetis_K_MK22F51212.MCUX_DEPRECATED_MACRO (*C macro*), 355
drivers_Kinetis_K_MK22F51212.MCUX_EXPERIMENTAL (*C macro*), 355
drivers_Kinetis_K_MK22F51212.MCUX_EXPERIMENTAL_MACRO (*C macro*), 355
drivers_Kinetis_K_MK22F51212.MCUX_MASK_INVERT_8 (*C macro*), 352
drivers_Kinetis_K_MK22F51212.MCUX_MASK_INVERT_16 (*C macro*), 352
drivers_Kinetis_K_MK22F51212.MCUX_MASK_INVERT_32 (*C macro*), 352
drivers_Kinetis_K_MK22F51212.MCUX_RAMFUNC (*C macro*), 355
drivers_Kinetis_K_MK22F51212.MCUX_REG_BIT_CLEAR8 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_BIT_CLEAR16 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_BIT_CLEAR32 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_BIT_GET8 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_BIT_GET16 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_BIT_GET32 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_BIT_SET8 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_BIT_SET16 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_BIT_SET32 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_MODIFY8 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_MODIFY16 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_MODIFY32 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_READ8 (*C macro*), 352
drivers_Kinetis_K_MK22F51212.MCUX_REG_READ16 (*C macro*), 352
drivers_Kinetis_K_MK22F51212.MCUX_REG_READ32 (*C macro*), 353
drivers_Kinetis_K_MK22F51212.MCUX_REG_WRITE8 (*C macro*), 352
drivers_Kinetis_K_MK22F51212.MCUX_REG_WRITE16 (*C macro*), 352
drivers_Kinetis_K_MK22F51212.MCUX_REG_WRITE32 (*C macro*), 352
drivers_Kinetis_K_MK22F51212.MCUX_SDK_SAI_ALLOW_NULL_FIFO_WATERMARK (*C macro*), 488
drivers_Kinetis_K_MK22F51212.MCUX_SDK_SAI_DISABLE_IMPLICIT_CHAN_CONFIG (*C macro*), 489
drivers_Kinetis_K_MK22F51212.MCUX_SDK_SAI_EDMA_RX_ENABLE_INTERNAL (*C macro*), 500
drivers_Kinetis_K_MK22F51212.MCUX_SDK_SAI_EDMA_TX_ENABLE_INTERNAL (*C macro*), 500
drivers_Kinetis_K_MK22F51212.MIN (*C macro*), 352
drivers_Kinetis_K_MK22F51212.MSDK_HAS_DWT_CYCCNT (*C macro*), 367
drivers_Kinetis_K_MK22F51212.MSDK_INVALID_IRQ_HANDLER (*C macro*), 367
drivers_Kinetis_K_MK22F51212.MSDK_REG_NONSECURE_ADDR (*C macro*), 367
drivers_Kinetis_K_MK22F51212.MSDK_REG_SECURE_ADDR (*C macro*), 367
drivers_Kinetis_K_MK22F51212.MSEC_TO_COUNT (*C macro*), 354
drivers_Kinetis_K_MK22F51212.osc_config_t (*C type*), 158

drivers_Kinetis_K_MK22F51212.osc_mode_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.OSC_SetCapLoad (*C function*), 168
drivers_Kinetis_K_MK22F51212.OSC_SetExtRefClkConfig (*C function*), 168
drivers_Kinetis_K_MK22F51212.oscer_config_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.oscerConfig (*C var*), 175
drivers_Kinetis_K_MK22F51212.oscsel (*C var*), 175
drivers_Kinetis_K_MK22F51212.pdb_adc_pretrigger_config_t (*C type*), 427
drivers_Kinetis_K_MK22F51212.pdb_adc_pretrigger_t (*C type*), 427
drivers_Kinetis_K_MK22F51212.pdb_adc_trigger_channel_t (*C type*), 427
drivers_Kinetis_K_MK22F51212.PDB_ClearADCPreTriggerStatusFlags (*C function*), 420
drivers_Kinetis_K_MK22F51212.PDB_ClearStatusFlags (*C function*), 418
drivers_Kinetis_K_MK22F51212.PDB_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.pdb_config_t (*C type*), 427
drivers_Kinetis_K_MK22F51212.pdb_dac_trigger_channel_t (*C type*), 427
drivers_Kinetis_K_MK22F51212.pdb_dac_trigger_config_t (*C type*), 427
drivers_Kinetis_K_MK22F51212.PDB_Deinit (*C function*), 417
drivers_Kinetis_K_MK22F51212.PDB_DisableInterrupts (*C function*), 418
drivers_Kinetis_K_MK22F51212.pdb_divider_multiplication_factor_t (*C type*), 426
drivers_Kinetis_K_MK22F51212.PDB_DoLoadValues (*C function*), 418
drivers_Kinetis_K_MK22F51212.PDB_DoSoftwareTrigger (*C function*), 418
drivers_Kinetis_K_MK22F51212.PDB_Enable (*C function*), 417
drivers_Kinetis_K_MK22F51212.PDB_EnableDMA (*C function*), 418
drivers_Kinetis_K_MK22F51212.PDB_EnableInterrupts (*C function*), 418
drivers_Kinetis_K_MK22F51212.PDB_EnablePulseOutTrigger (*C function*), 420
drivers_Kinetis_K_MK22F51212.PDB_GetADCPreTriggerStatusFlags (*C function*), 419
drivers_Kinetis_K_MK22F51212.PDB_GetCounterValue (*C function*), 419
drivers_Kinetis_K_MK22F51212.PDB_GetDefaultConfig (*C function*), 417
drivers_Kinetis_K_MK22F51212.PDB_GetStatusFlags (*C function*), 418
drivers_Kinetis_K_MK22F51212.PDB_Init (*C function*), 417
drivers_Kinetis_K_MK22F51212.pdb_load_value_mode_t (*C type*), 426
drivers_Kinetis_K_MK22F51212.pdb_prescaler_divider_t (*C type*), 426
drivers_Kinetis_K_MK22F51212.pdb_pulse_out_channel_mask_t (*C type*), 427
drivers_Kinetis_K_MK22F51212.pdb_pulse_out_trigger_channel_t (*C type*), 427
drivers_Kinetis_K_MK22F51212.PDB_SetADCPreTriggerConfig (*C function*), 419
drivers_Kinetis_K_MK22F51212.PDB_SetADCPreTriggerDelayValue (*C function*), 419
drivers_Kinetis_K_MK22F51212.PDB_SetCounterDelayValue (*C function*), 419
drivers_Kinetis_K_MK22F51212.PDB_SetDACTriggerConfig (*C function*), 420
drivers_Kinetis_K_MK22F51212.PDB_SetDACTriggerIntervalValue (*C function*), 420
drivers_Kinetis_K_MK22F51212.PDB_SetModulusValue (*C function*), 418
drivers_Kinetis_K_MK22F51212.PDB_SetPulseOutTriggerDelayValue (*C function*), 420
drivers_Kinetis_K_MK22F51212.pdb_trigger_input_source_t (*C type*), 426
drivers_Kinetis_K_MK22F51212.pflash_prot_status_t (*C type*), 291
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kClearFlags (*C enumerator*), 345
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_ChannelPriorityErrorFlag (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_DestinationAddressErrorFlag (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_DestinationBusErrorFlag (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_DestinationOffsetErrorFlag (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_DoneFlag (*C enumerator*), 240
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_ErrorChannelFlag (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_ErrorFlag (*C enumerator*), 240
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_InterruptFlag (*C enumerator*), 240

drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_NbytesErrorFlag (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_ScatterGatherErrorFlag (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_SourceAddressErrorFlag (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_SourceBusErrorFlag (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_SourceOffsetErrorFlag (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_TransferCanceledFlag (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kEDMA_ValidFlag (*C enumerator*), 241
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kIrqFlags (*C enumerator*), 345
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCG_IrclkEnable (*C enumerator*), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCG_IrclkEnableInStop (*C enumerator*), 157
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCG_Osc0InitFlag (*C enumerator*), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCG_Osc0LostFlag (*C enumerator*), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCG_Pll0LockFlag (*C enumerator*), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCG_Pll0LostFlag (*C enumerator*), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCG_PllEnableIndependent (*C enumerator*), 157
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCG_PllEnableInStop (*C enumerator*), 157
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCG_RtcOscLostFlag (*C enumerator*), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCM_CacheWriteBuffer (*C enumerator*), 411
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCM_FPUDivideByZero (*C enumerator*), 412
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCM_FPUInexact (*C enumerator*), 412
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCM_FPUInputDenormalInterrupt (*C enumerator*), 412
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCM_FPUInvalidOperation (*C enumerator*), 412
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCM_FPUOverflow (*C enumerator*), 412
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCM_FPUUnderflow (*C enumerator*), 412
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kMCM_ParityError (*C enumerator*), 412
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_Channel0Mask (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_Channel1Mask (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_Channel2Mask (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_Channel3Mask (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_Channel4Mask (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_Channel5Mask (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_Channel6Mask (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_Channel7Mask (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_FIFOErrorFlag (*C enumerator*), 484
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_FIFOErrorInterruptEnable (*C enumerator*), 483
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_FIFORequestDMAEnable (*C enumerator*), 484
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_FIFORequestFlag (*C enumerator*), 484
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_FIFORequestInterruptEnable (*C enumerator*), 483
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_FIFOWarningDMAEnable (*C enu-*

merator), 483
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_FIFOWarningFlag (*C enumerator*), 484
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_FIFOWarningInterruptEnable (*C enumerator*), 483
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_SyncErrorFlag (*C enumerator*), 484
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_SyncErrorInterruptEnable (*C enumerator*), 483
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_WordStartFlag (*C enumerator*), 484
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kSAI_WordStartInterruptEnable (*C enumerator*), 483
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_Busy (*C enumerator*), 364
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_DSPI_Busy (*C enumerator*), 204
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_DSPI_Error (*C enumerator*), 205
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_DSPI_Idle (*C enumerator*), 205
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_DSPI_OutOfRange (*C enumerator*), 205
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_EDMA_Busy (*C enumerator*), 242
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_EDMA_QueueFull (*C enumerator*), 242
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_Fail (*C enumerator*), 364
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_AccessError (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_AddressError (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_AlignmentError (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_CommandFailure (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_CommandNotSupported (*C enumerator*), 260
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_CommandOperationInProgress (*C enumerator*), 260
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_EraseKeyError (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_ExecuteInRamFunctionNotReady (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_InvalidArgument (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_InvalidPropertyValue (*C enumerator*), 260
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_InvalidSpeculationOption (*C enumerator*), 260
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_PartitionStatusUpdateFailure (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_ProtectionViolation (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_ReadOnlyProperty (*C enumerator*), 260
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_RecoverFlexramAsEepromError (*C enumerator*), 260
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_RecoverFlexramAsRamError (*C enumerator*), 260
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_RegionExecuteOnly (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_SetFlexramAsEepromError (*C enumerator*), 260

drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_SetFlexramAsRamError (*C enumerator*), 260
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_SizeError (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_Success (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_SwapIndicatorAddressError (*C enumerator*), 260
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_SwapSystemNotInUninitialized (*C enumerator*), 260
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_FTFx_UnknownProperty (*C enumerator*), 259
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_I2C_Addr_Nak (*C enumerator*), 342
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_I2C_ArbitrationLost (*C enumerator*), 342
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_I2C_Busy (*C enumerator*), 342
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_I2C_Idle (*C enumerator*), 342
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_I2C_Nak (*C enumerator*), 342
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_I2C_Timeout (*C enumerator*), 342
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_InvalidArgument (*C enumerator*), 364
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_BaudrateNotSupport (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_Error (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_FlagCannotClearManually (*C enumerator*), 398
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_FramingError (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_IdleLineDetected (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_NoiseError (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_ParityError (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_RxBusy (*C enumerator*), 398
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_RxHardwareOverrun (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_RxIdle (*C enumerator*), 398
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_RxRingBufferOverrun (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_RxWatermarkTooLarge (*C enumerator*), 398
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_Timeout (*C enumerator*), 399
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_TxBusy (*C enumerator*), 398
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_TxIdle (*C enumerator*), 398
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_LPUART_TxWatermarkTooLarge (*C enumerator*), 398
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_MCG_AtmBusClockInvalid (*C enumerator*), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_MCG_AtmDesiredFreqInvalid (*C*

enumerator), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_MCG_AtmHardwareFail (*C enumerator*), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_MCG_AtmIrcUsed (*C enumerator*), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_MCG_ModeInvalid (*C enumerator*), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_MCG_ModeUnreachable (*C enumerator*), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_MCG_SourceUsed (*C enumerator*), 156
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_NoData (*C enumerator*), 364
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_NoTransferInProgress (*C enumerator*), 364
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_OutOfRange (*C enumerator*), 364
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_ReadOnly (*C enumerator*), 364
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SAI_QueueFull (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SAI_RxBusy (*C enumerator*), 479
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SAI_RxError (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SAI_RxIdle (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SAI_TxBusy (*C enumerator*), 479
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SAI_TxError (*C enumerator*), 479
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SAI_TxIdle (*C enumerator*), 480
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SMARTCARD_CardNotActivated (*C enumerator*), 503
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SMARTCARD_Initialized (*C enumerator*), 503
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SMARTCARD_InvalidInput (*C enumerator*), 503
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SMARTCARD_NoTransferInProgress (*C enumerator*), 503
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SMARTCARD_OtherError (*C enumerator*), 504
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SMARTCARD_PhyInitialized (*C enumerator*), 503
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SMARTCARD_RxBusy (*C enumerator*), 503
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SMARTCARD_Success (*C enumerator*), 503
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SMARTCARD_Timeout (*C enumerator*), 503
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SMARTCARD_TxBusy (*C enumerator*), 503
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_SMC_StopAbort (*C enumerator*), 523
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_Success (*C enumerator*), 364
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_Timeout (*C enumerator*), 364
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_BaudrateNotSupport (*C enumerator*), 536
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_Error (*C enumerator*), 535
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_FlagCannotClearManually (*C enumerator*), 535
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_FramingError (*C enumer-*

ator), 536
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_IdleLineDetected (C enumerator), 536
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_NoiseError (C enumerator), 536
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_ParityError (C enumerator), 536
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_RxBusy (C enumerator), 535
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_RxHardwareOverrun (C enumerator), 535
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_RxIdle (C enumerator), 535
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_RxRingBufferOverrun (C enumerator), 535
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_RxWatermarkTooLarge (C enumerator), 535
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_Timeout (C enumerator), 536
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_TxBusy (C enumerator), 535
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_TxIdle (C enumerator), 535
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kStatus_UART_TxWatermarkTooLarge (C enumerator), 535
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_FramingErrorFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_IdleLineFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_LinBreakFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_NoiseErrorFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_NoiseErrorInRxDataRegFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_ParityErrorFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_ParityErrorInRxDataRegFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_RxActiveEdgeFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_RxActiveFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_RxDataRegFullFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_RxFifoEmptyFlag (C enumerator), 539
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_RxFifoOverflowFlag (C enumerator), 539
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_RxFifoUnderflowFlag (C enumerator), 539
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_RxOverrunFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_TransmissionCompleteFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_TxDataRegEmptyFlag (C enumerator), 538
drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_TxFifoEmptyFlag (C enumerator), 539

drivers_Kinetis_K_MK22F51212.PhonyNameDueToError.kUART_TxFifoOverflowFlag (*C enumerator*), 539
drivers_Kinetis_K_MK22F51212.pit_chnl_t (*C type*), 432
drivers_Kinetis_K_MK22F51212.PIT_ClearStatusFlags (*C function*), 430
drivers_Kinetis_K_MK22F51212.PIT_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.pit_config_t (*C type*), 432
drivers_Kinetis_K_MK22F51212.PIT_Deinit (*C function*), 429
drivers_Kinetis_K_MK22F51212.PIT_DisableInterrupts (*C function*), 429
drivers_Kinetis_K_MK22F51212.PIT_EnableInterrupts (*C function*), 429
drivers_Kinetis_K_MK22F51212.PIT_GetCurrentTimerCount (*C function*), 430
drivers_Kinetis_K_MK22F51212.PIT_GetDefaultConfig (*C function*), 429
drivers_Kinetis_K_MK22F51212.PIT_GetEnabledInterrupts (*C function*), 430
drivers_Kinetis_K_MK22F51212.PIT_GetLifetimeTimerCount (*C function*), 432
drivers_Kinetis_K_MK22F51212.PIT_GetStatusFlags (*C function*), 430
drivers_Kinetis_K_MK22F51212.PIT_Init (*C function*), 429
drivers_Kinetis_K_MK22F51212.pit_interrupt_enable_t (*C type*), 432
drivers_Kinetis_K_MK22F51212.PIT_SetTimerChainMode (*C function*), 429
drivers_Kinetis_K_MK22F51212.PIT_SetTimerPeriod (*C function*), 430
drivers_Kinetis_K_MK22F51212.PIT_StartTimer (*C function*), 431
drivers_Kinetis_K_MK22F51212.pit_status_flags_t (*C type*), 432
drivers_Kinetis_K_MK22F51212.PIT_StopTimer (*C function*), 431
drivers_Kinetis_K_MK22F51212.pll0Config (*C var*), 176
drivers_Kinetis_K_MK22F51212.pllFllSel (*C var*), 174
drivers_Kinetis_K_MK22F51212.pmc_bandgap_buffer_config_t (*C type*), 438
drivers_Kinetis_K_MK22F51212.pmc_bandgap_buffer_drive_select_t (*C type*), 437
drivers_Kinetis_K_MK22F51212.PMC_ClearHighVoltDetectFlag (*C function*), 435
drivers_Kinetis_K_MK22F51212.PMC_ClearLowVoltDetectFlag (*C function*), 434
drivers_Kinetis_K_MK22F51212.PMC_ClearLowVoltWarningFlag (*C function*), 434
drivers_Kinetis_K_MK22F51212.PMC_ClearPeriphIOIsolationFlag (*C function*), 435
drivers_Kinetis_K_MK22F51212.PMC_ConfigureBandgapBuffer (*C function*), 435
drivers_Kinetis_K_MK22F51212.PMC_ConfigureHighVoltDetect (*C function*), 434
drivers_Kinetis_K_MK22F51212.PMC_ConfigureLowVoltDetect (*C function*), 433
drivers_Kinetis_K_MK22F51212.PMC_ConfigureLowVoltWarning (*C function*), 434
drivers_Kinetis_K_MK22F51212.PMC_GetHighVoltDetectFlag (*C function*), 434
drivers_Kinetis_K_MK22F51212.PMC_GetLowVoltDetectFlag (*C function*), 433
drivers_Kinetis_K_MK22F51212.PMC_GetLowVoltWarningFlag (*C function*), 434
drivers_Kinetis_K_MK22F51212.PMC_GetParam (*C function*), 433
drivers_Kinetis_K_MK22F51212.PMC_GetPeriphIOIsolationFlag (*C function*), 435
drivers_Kinetis_K_MK22F51212.PMC_GetVersionId (*C function*), 433
drivers_Kinetis_K_MK22F51212.pmc_high_volt_detect_config_t (*C type*), 437
drivers_Kinetis_K_MK22F51212.pmc_high_volt_detect_volt_select_t (*C type*), 437
drivers_Kinetis_K_MK22F51212.PMC_IsRegulatorInRunRegulation (*C function*), 435
drivers_Kinetis_K_MK22F51212.pmc_low_volt_detect_config_t (*C type*), 437
drivers_Kinetis_K_MK22F51212.pmc_low_volt_detect_volt_select_t (*C type*), 437
drivers_Kinetis_K_MK22F51212.pmc_low_volt_warning_config_t (*C type*), 437
drivers_Kinetis_K_MK22F51212.pmc_low_volt_warning_volt_select_t (*C type*), 437
drivers_Kinetis_K_MK22F51212.pmc_param_t (*C type*), 437
drivers_Kinetis_K_MK22F51212.pmc_version_id_t (*C type*), 437
drivers_Kinetis_K_MK22F51212.pmc_vlp_freq_mode_t (*C type*), 437
drivers_Kinetis_K_MK22F51212.PORT_ClearPinsInterruptFlags (*C function*), 443
drivers_Kinetis_K_MK22F51212.PORT_CLOCKS (*C macro*), 162
drivers_Kinetis_K_MK22F51212.port_digital_filter_clock_source_t (*C type*), 447
drivers_Kinetis_K_MK22F51212.port_digital_filter_config_t (*C type*), 447
drivers_Kinetis_K_MK22F51212.PORT_EnablePinsDigitalFilter (*C function*), 441
drivers_Kinetis_K_MK22F51212.PORT_GetPinsInterruptFlags (*C function*), 442
drivers_Kinetis_K_MK22F51212.port_interrupt_t (*C type*), 447
drivers_Kinetis_K_MK22F51212.port_mux_t (*C type*), 447
drivers_Kinetis_K_MK22F51212.port_pin_config_t (*C type*), 447

drivers_Kinetis_K_MK22F51212.PORT_SetDigitalFilterConfig (*C function*), 441
drivers_Kinetis_K_MK22F51212.PORT_SetMultipleInterruptPinsConfig (*C function*), 440
drivers_Kinetis_K_MK22F51212.PORT_SetMultiplePinsConfig (*C function*), 440
drivers_Kinetis_K_MK22F51212.PORT_SetPinConfig (*C function*), 439
drivers_Kinetis_K_MK22F51212.PORT_SetPinDriveStrength (*C function*), 442
drivers_Kinetis_K_MK22F51212.PORT_SetPinInterruptConfig (*C function*), 442
drivers_Kinetis_K_MK22F51212.PORT_SetPinMux (*C function*), 441
drivers_Kinetis_K_MK22F51212.prdiv (*C var*), 175
drivers_Kinetis_K_MK22F51212.RAMFUNCTION_SECTION_CODE (*C macro*), 355
drivers_Kinetis_K_MK22F51212.rcm_boot_rom_config_t (*C type*), 453
drivers_Kinetis_K_MK22F51212.RCM_ClearBootRomSource (*C function*), 450
drivers_Kinetis_K_MK22F51212.RCM_ClearStickyResetSources (*C function*), 449
drivers_Kinetis_K_MK22F51212.RCM_ConfigureResetPinFilter (*C function*), 449
drivers_Kinetis_K_MK22F51212.RCM_GetBootRomSource (*C function*), 450
drivers_Kinetis_K_MK22F51212.RCM_GetEasyPortModePinStatus (*C function*), 449
drivers_Kinetis_K_MK22F51212.RCM_GetPreviousResetSources (*C function*), 448
drivers_Kinetis_K_MK22F51212.RCM_GetResetSourceImplementedStatus (*C function*), 448
drivers_Kinetis_K_MK22F51212.RCM_GetStickyResetSources (*C function*), 449
drivers_Kinetis_K_MK22F51212.RCM_GetVersionId (*C function*), 448
drivers_Kinetis_K_MK22F51212.rcm_interrupt_enable_t (*C type*), 453
drivers_Kinetis_K_MK22F51212.rcm_reset_delay_t (*C type*), 453
drivers_Kinetis_K_MK22F51212.rcm_reset_pin_filter_config_t (*C type*), 453
drivers_Kinetis_K_MK22F51212.rcm_reset_source_t (*C type*), 453
drivers_Kinetis_K_MK22F51212.rcm_run_wait_filter_mode_t (*C type*), 453
drivers_Kinetis_K_MK22F51212.RCM_SetForceBootRomSource (*C function*), 450
drivers_Kinetis_K_MK22F51212.RCM_SetSystemResetInterruptConfig (*C function*), 450
drivers_Kinetis_K_MK22F51212.rcm_version_id_t (*C type*), 453
drivers_Kinetis_K_MK22F51212.RNGA_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.RNGA_Deinit (*C function*), 455
drivers_Kinetis_K_MK22F51212.RNGA_GetMode (*C function*), 455
drivers_Kinetis_K_MK22F51212.RNGA_GetRandomData (*C function*), 455
drivers_Kinetis_K_MK22F51212.RNGA_Init (*C function*), 455
drivers_Kinetis_K_MK22F51212.rnga_mode_t (*C type*), 455
drivers_Kinetis_K_MK22F51212.RNGA_Seed (*C function*), 455
drivers_Kinetis_K_MK22F51212.RNGA_SetMode (*C function*), 455
drivers_Kinetis_K_MK22F51212.RTC_ClearStatusFlags (*C function*), 458
drivers_Kinetis_K_MK22F51212.RTC_CLOCKS (*C macro*), 162
drivers_Kinetis_K_MK22F51212.rtc_config_t (*C type*), 462
drivers_Kinetis_K_MK22F51212.rtc_datetime_t (*C type*), 462
drivers_Kinetis_K_MK22F51212.RTC_Deinit (*C function*), 456
drivers_Kinetis_K_MK22F51212.RTC_DisableInterrupts (*C function*), 457
drivers_Kinetis_K_MK22F51212.RTC_EnableClockOutput (*C function*), 463
drivers_Kinetis_K_MK22F51212.RTC_EnableInterrupts (*C function*), 457
drivers_Kinetis_K_MK22F51212.RTC_EnableLPOClock (*C function*), 458
drivers_Kinetis_K_MK22F51212.RTC_EnableOscillatorClock (*C function*), 458
drivers_Kinetis_K_MK22F51212.RTC_EnableWakeUpPin (*C function*), 462
drivers_Kinetis_K_MK22F51212.RTC_GetAlarm (*C function*), 457
drivers_Kinetis_K_MK22F51212.RTC_GetDatetime (*C function*), 456
drivers_Kinetis_K_MK22F51212.RTC_GetDefaultConfig (*C function*), 456
drivers_Kinetis_K_MK22F51212.RTC_GetEnabledInterrupts (*C function*), 457
drivers_Kinetis_K_MK22F51212.RTC_GetMonotonicCounter (*C function*), 459
drivers_Kinetis_K_MK22F51212.RTC_GetStatusFlags (*C function*), 457
drivers_Kinetis_K_MK22F51212.RTC_GetTamperTimeSeconds (*C function*), 462
drivers_Kinetis_K_MK22F51212.RTC_IncrementMonotonicCounter (*C function*), 459
drivers_Kinetis_K_MK22F51212.RTC_Init (*C function*), 456
drivers_Kinetis_K_MK22F51212.rtc_interrupt_enable_t (*C type*), 461
drivers_Kinetis_K_MK22F51212.rtc_osc_cap_load_t (*C type*), 462
drivers_Kinetis_K_MK22F51212.rtc_pin_config_t (*C type*), 462

drivers_Kinetis_K_MK22F51212.RTC_Reset (*C function*), 462
drivers_Kinetis_K_MK22F51212.RTC_SetAlarm (*C function*), 457
drivers_Kinetis_K_MK22F51212.RTC_SetClockSource (*C function*), 458
drivers_Kinetis_K_MK22F51212.RTC_SetDatetime (*C function*), 456
drivers_Kinetis_K_MK22F51212.RTC_SetMonotonicCounter (*C function*), 459
drivers_Kinetis_K_MK22F51212.RTC_SetOscCapLoad (*C function*), 462
drivers_Kinetis_K_MK22F51212.RTC_SetTimerSecondsInterruptFrequency (*C function*), 463
drivers_Kinetis_K_MK22F51212.RTC_StartTimer (*C function*), 458
drivers_Kinetis_K_MK22F51212.rtc_status_flags_t (*C type*), 462
drivers_Kinetis_K_MK22F51212.RTC_StopTimer (*C function*), 459
drivers_Kinetis_K_MK22F51212.rtc_timer_seconds_interrupt_frequency_t (*C type*), 462
drivers_Kinetis_K_MK22F51212.s_lpuartHandle (*C var*), 404
drivers_Kinetis_K_MK22F51212.s_lpuartIsr (*C var*), 405
drivers_Kinetis_K_MK22F51212.s_lpuartTxIRQ (*C var*), 404
drivers_Kinetis_K_MK22F51212.s_uartHandle (*C var*), 539
drivers_Kinetis_K_MK22F51212.s_uartIRQ (*C var*), 539
drivers_Kinetis_K_MK22F51212.s_uartIsr (*C var*), 539
drivers_Kinetis_K_MK22F51212.sai_bclk_source_t (*C type*), 487
drivers_Kinetis_K_MK22F51212.sai_bit_clock_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.sai_clock_polarity_t (*C type*), 487
drivers_Kinetis_K_MK22F51212.SAI_CLOCKS (*C macro*), 162
drivers_Kinetis_K_MK22F51212.sai_config_t (*C type*), 487
drivers_Kinetis_K_MK22F51212.sai_data_order_t (*C type*), 487
drivers_Kinetis_K_MK22F51212.sai_data_pin_state_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.SAI_Deinit (*C function*), 465
drivers_Kinetis_K_MK22F51212.SAI_DriverIRQHandler (*C function*), 479
drivers_Kinetis_K_MK22F51212.sai_edma_callback_t (*C type*), 500
drivers_Kinetis_K_MK22F51212.sai_edma_handle (*C struct*), 500
drivers_Kinetis_K_MK22F51212.sai_edma_handle_t (*C type*), 500
drivers_Kinetis_K_MK22F51212.sai_edma_handle.bytesPerFrame (*C var*), 500
drivers_Kinetis_K_MK22F51212.sai_edma_handle.callback (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_handle.channel (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_handle.channelMask (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_handle.channelNums (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_handle.count (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_handle.dmaHandle (*C var*), 500
drivers_Kinetis_K_MK22F51212.sai_edma_handle.interleaveType (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_handle.nbytes (*C var*), 500
drivers_Kinetis_K_MK22F51212.sai_edma_handle.queueDriver (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_handle.queueUser (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_handle.saiQueue (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_handle.state (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_handle.tcd (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_handle.transferSize (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_handle.userData (*C var*), 501
drivers_Kinetis_K_MK22F51212.sai_edma_interleave_t (*C type*), 500
drivers_Kinetis_K_MK22F51212.sai_fifo_combine_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.sai_fifo_packing_t (*C type*), 487
drivers_Kinetis_K_MK22F51212.sai_fifo_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.sai_frame_sync_len_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.sai_frame_sync_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.SAI_GetClassicI2SConfig (*C function*), 468
drivers_Kinetis_K_MK22F51212.SAI_GetDSPConfig (*C function*), 469
drivers_Kinetis_K_MK22F51212.SAI_GetLeftJustifiedConfig (*C function*), 468
drivers_Kinetis_K_MK22F51212.SAI_GetRightJustifiedConfig (*C function*), 468
drivers_Kinetis_K_MK22F51212.SAI_GetTDMConfig (*C function*), 469
drivers_Kinetis_K_MK22F51212.sai_handle_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.SAI_Init (*C function*), 465

drivers_Kinetis_K_MK22F51212.sai_master_clock_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.sai_master_slave_t (*C type*), 487
drivers_Kinetis_K_MK22F51212.sai_mclk_source_t (*C type*), 487
drivers_Kinetis_K_MK22F51212.sai_mono_stereo_t (*C type*), 487
drivers_Kinetis_K_MK22F51212.sai_protocol_t (*C type*), 487
drivers_Kinetis_K_MK22F51212.SAI_ReadBlocking (*C function*), 475
drivers_Kinetis_K_MK22F51212.SAI_ReadData (*C function*), 476
drivers_Kinetis_K_MK22F51212.SAI_ReadMultiChannelBlocking (*C function*), 475
drivers_Kinetis_K_MK22F51212.sai_reset_type_t (*C type*), 487
drivers_Kinetis_K_MK22F51212.SAI_RxClearStatusFlags (*C function*), 470
drivers_Kinetis_K_MK22F51212.SAI_RxDisableInterrupts (*C function*), 473
drivers_Kinetis_K_MK22F51212.SAI_RxEnable (*C function*), 465
drivers_Kinetis_K_MK22F51212.SAI_RxEnableDMA (*C function*), 474
drivers_Kinetis_K_MK22F51212.SAI_RxEnableInterrupts (*C function*), 473
drivers_Kinetis_K_MK22F51212.SAI_RxGetDataRegisterAddress (*C function*), 474
drivers_Kinetis_K_MK22F51212.SAI_RxGetStatusFlag (*C function*), 470
drivers_Kinetis_K_MK22F51212.SAI_RxReset (*C function*), 465
drivers_Kinetis_K_MK22F51212.SAI_RxSetBitclockConfig (*C function*), 467
drivers_Kinetis_K_MK22F51212.SAI_RxSetBitClockDirection (*C function*), 466
drivers_Kinetis_K_MK22F51212.SAI_RxSetBitClockPolarity (*C function*), 471
drivers_Kinetis_K_MK22F51212.SAI_RxSetBitClockRate (*C function*), 466
drivers_Kinetis_K_MK22F51212.SAI_RxSetChannelFIFOMask (*C function*), 471
drivers_Kinetis_K_MK22F51212.SAI_RxSetConfig (*C function*), 468
drivers_Kinetis_K_MK22F51212.SAI_RxSetDataOrder (*C function*), 471
drivers_Kinetis_K_MK22F51212.SAI_RxSetFifoConfig (*C function*), 467
drivers_Kinetis_K_MK22F51212.SAI_RxSetFIFOErrorContinue (*C function*), 472
drivers_Kinetis_K_MK22F51212.SAI_RxSetFIFOPacking (*C function*), 472
drivers_Kinetis_K_MK22F51212.SAI_RxSetFrameSyncConfig (*C function*), 467
drivers_Kinetis_K_MK22F51212.SAI_RxSetFrameSyncDirection (*C function*), 466
drivers_Kinetis_K_MK22F51212.SAI_RxSetFrameSyncPolarity (*C function*), 472
drivers_Kinetis_K_MK22F51212.SAI_RxSetSerialDataConfig (*C function*), 468
drivers_Kinetis_K_MK22F51212.SAI_RxSoftwareReset (*C function*), 470
drivers_Kinetis_K_MK22F51212.sai_sample_rate_t (*C type*), 487
drivers_Kinetis_K_MK22F51212.sai_serial_data_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.SAI_SetMasterClockConfig (*C function*), 467
drivers_Kinetis_K_MK22F51212.sai_sync_mode_t (*C type*), 487
drivers_Kinetis_K_MK22F51212.sai_transceiver_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.sai_transceiver_type_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.sai_transfer_callback_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.sai_transfer_format_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.sai_transfer_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.SAI_TransferAbortReceive (*C function*), 478
drivers_Kinetis_K_MK22F51212.SAI_TransferAbortReceiveEDMA (*C function*), 499
drivers_Kinetis_K_MK22F51212.SAI_TransferAbortSend (*C function*), 478
drivers_Kinetis_K_MK22F51212.SAI_TransferAbortSendEDMA (*C function*), 498
drivers_Kinetis_K_MK22F51212.SAI_TransferGetReceiveCount (*C function*), 478
drivers_Kinetis_K_MK22F51212.SAI_TransferGetReceiveCountEDMA (*C function*), 499
drivers_Kinetis_K_MK22F51212.SAI_TransferGetSendCount (*C function*), 478
drivers_Kinetis_K_MK22F51212.SAI_TransferGetSendCountEDMA (*C function*), 499
drivers_Kinetis_K_MK22F51212.SAI_TransferGetValidTransferSlotsEDMA (*C function*), 499
drivers_Kinetis_K_MK22F51212.SAI_TransferReceiveEDMA (*C function*), 497
drivers_Kinetis_K_MK22F51212.SAI_TransferReceiveLoopEDMA (*C function*), 498
drivers_Kinetis_K_MK22F51212.SAI_TransferReceiveNonBlocking (*C function*), 477
drivers_Kinetis_K_MK22F51212.SAI_TransferRxCreateHandle (*C function*), 476
drivers_Kinetis_K_MK22F51212.SAI_TransferRxCreateHandleEDMA (*C function*), 495
drivers_Kinetis_K_MK22F51212.SAI_TransferRxHandleIRQ (*C function*), 479
drivers_Kinetis_K_MK22F51212.SAI_TransferRxSetConfig (*C function*), 477
drivers_Kinetis_K_MK22F51212.SAI_TransferRxSetConfigEDMA (*C function*), 496

drivers_Kinetis_K_MK22F51212.SAI_TransferSendEDMA (*C function*), 496
drivers_Kinetis_K_MK22F51212.SAI_TransferSendLoopEDMA (*C function*), 497
drivers_Kinetis_K_MK22F51212.SAI_TransferSendNonBlocking (*C function*), 477
drivers_Kinetis_K_MK22F51212.SAI_TransferSetInterleaveType (*C function*), 495
drivers_Kinetis_K_MK22F51212.SAI_TransferTerminateReceive (*C function*), 479
drivers_Kinetis_K_MK22F51212.SAI_TransferTerminateReceiveEDMA (*C function*), 498
drivers_Kinetis_K_MK22F51212.SAI_TransferTerminateSend (*C function*), 478
drivers_Kinetis_K_MK22F51212.SAI_TransferTerminateSendEDMA (*C function*), 498
drivers_Kinetis_K_MK22F51212.SAI_TransferTxCreateHandle (*C function*), 476
drivers_Kinetis_K_MK22F51212.SAI_TransferTxCreateHandleEDMA (*C function*), 494
drivers_Kinetis_K_MK22F51212.SAI_TransferTxHandleIRQ (*C function*), 479
drivers_Kinetis_K_MK22F51212.SAI_TransferTxSetConfig (*C function*), 476
drivers_Kinetis_K_MK22F51212.SAI_TransferTxSetConfigEDMA (*C function*), 495
drivers_Kinetis_K_MK22F51212.SAI_TxClearStatusFlags (*C function*), 470
drivers_Kinetis_K_MK22F51212.SAI_TxDisableInterrupts (*C function*), 473
drivers_Kinetis_K_MK22F51212.SAI_TxEnable (*C function*), 465
drivers_Kinetis_K_MK22F51212.SAI_TxEnableDMA (*C function*), 473
drivers_Kinetis_K_MK22F51212.SAI_TxEnableInterrupts (*C function*), 472
drivers_Kinetis_K_MK22F51212.SAI_TxGetDataRegisterAddress (*C function*), 474
drivers_Kinetis_K_MK22F51212.SAI_TxGetStatusFlag (*C function*), 469
drivers_Kinetis_K_MK22F51212.SAI_TxReset (*C function*), 465
drivers_Kinetis_K_MK22F51212.SAI_TxSetBitclockConfig (*C function*), 466
drivers_Kinetis_K_MK22F51212.SAI_TxSetBitClockDirection (*C function*), 465
drivers_Kinetis_K_MK22F51212.SAI_TxSetBitClockPolarity (*C function*), 471
drivers_Kinetis_K_MK22F51212.SAI_TxSetBitClockRate (*C function*), 466
drivers_Kinetis_K_MK22F51212.SAI_TxSetChannelFIFOMask (*C function*), 471
drivers_Kinetis_K_MK22F51212.SAI_TxSetConfig (*C function*), 468
drivers_Kinetis_K_MK22F51212.SAI_TxSetDataOrder (*C function*), 471
drivers_Kinetis_K_MK22F51212.SAI_TxSetFifoConfig (*C function*), 467
drivers_Kinetis_K_MK22F51212.SAI_TxSetFIFOErrorContinue (*C function*), 472
drivers_Kinetis_K_MK22F51212.SAI_TxSetFIFOPacking (*C function*), 472
drivers_Kinetis_K_MK22F51212.SAI_TxSetFrameSyncConfig (*C function*), 467
drivers_Kinetis_K_MK22F51212.SAI_TxSetFrameSyncDirection (*C function*), 466
drivers_Kinetis_K_MK22F51212.SAI_TxSetFrameSyncPolarity (*C function*), 471
drivers_Kinetis_K_MK22F51212.SAI_TxSetSerialDataConfig (*C function*), 468
drivers_Kinetis_K_MK22F51212.SAI_TxSoftwareReset (*C function*), 470
drivers_Kinetis_K_MK22F51212.sai_word_width_t (*C type*), 488
drivers_Kinetis_K_MK22F51212.SAI_WriteBlocking (*C function*), 474
drivers_Kinetis_K_MK22F51212.SAI_WriteData (*C function*), 475
drivers_Kinetis_K_MK22F51212.SAI_WriteMultiChannelBlocking (*C function*), 475
drivers_Kinetis_K_MK22F51212.SAI_XFER_QUEUE_SIZE (*C macro*), 489
drivers_Kinetis_K_MK22F51212.SBR_CAL_ADJUST_D1_T0 (*C macro*), 517
drivers_Kinetis_K_MK22F51212.SBR_CAL_ADJUST_D1_T1 (*C macro*), 517
drivers_Kinetis_K_MK22F51212.SBR_CAL_ADJUST_D2_T0 (*C macro*), 517
drivers_Kinetis_K_MK22F51212.SBR_CAL_ADJUST_D2_T1 (*C macro*), 518
drivers_Kinetis_K_MK22F51212.SBR_CAL_ADJUST_D4_T0 (*C macro*), 517
drivers_Kinetis_K_MK22F51212.SBR_CAL_ADJUST_D4_T1 (*C macro*), 518
drivers_Kinetis_K_MK22F51212.SDK_ALIGN (*C macro*), 354
drivers_Kinetis_K_MK22F51212.SDK_ATOMIC_LOCAL_ADD (*C macro*), 353
drivers_Kinetis_K_MK22F51212.SDK_ATOMIC_LOCAL_CLEAR (*C macro*), 353
drivers_Kinetis_K_MK22F51212.SDK_ATOMIC_LOCAL_CLEAR_AND_SET (*C macro*), 353
drivers_Kinetis_K_MK22F51212.SDK_ATOMIC_LOCAL_COMPARE_AND_SET (*C macro*), 353
drivers_Kinetis_K_MK22F51212.SDK_ATOMIC_LOCAL_SET (*C macro*), 353
drivers_Kinetis_K_MK22F51212.SDK_ATOMIC_LOCAL_SUB (*C macro*), 353
drivers_Kinetis_K_MK22F51212.SDK_ATOMIC_LOCAL_TEST_AND_SET (*C macro*), 354
drivers_Kinetis_K_MK22F51212.SDK_ATOMIC_LOCAL_TOGGLE (*C macro*), 353
drivers_Kinetis_K_MK22F51212.SDK_DelayAtLeastUs (*C function*), 365

drivers_Kinetis_K_MK22F51212.SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY (*C macro*), 162
drivers_Kinetis_K_MK22F51212.SDK_Free (*C function*), 364
drivers_Kinetis_K_MK22F51212.SDK_IS_ALIGNED (*C macro*), 354
drivers_Kinetis_K_MK22F51212.SDK_ISR_EXIT_BARRIER (*C macro*), 354
drivers_Kinetis_K_MK22F51212.SDK_Malloc (*C function*), 364
drivers_Kinetis_K_MK22F51212.SDK_SIZEALIGN (*C macro*), 354
drivers_Kinetis_K_MK22F51212.SDK_SIZEALIGN_DOWN (*C macro*), 354
drivers_Kinetis_K_MK22F51212.SDK_SIZEALIGN_UP (*C macro*), 354
drivers_Kinetis_K_MK22F51212.sim_clock_config_t (*C type*), 158
drivers_Kinetis_K_MK22F51212.SIM_GetUniqueId (*C function*), 502
drivers_Kinetis_K_MK22F51212.SIM_SetFlashMode (*C function*), 502
drivers_Kinetis_K_MK22F51212.SIM_SetUsbVoltRegulatorEnableMode (*C function*), 502
drivers_Kinetis_K_MK22F51212.sim_uid_t (*C type*), 502
drivers_Kinetis_K_MK22F51212.SMARTCARD_ATR_DURATION_ADJUSTMENT (*C macro*), 514
drivers_Kinetis_K_MK22F51212.smartcard_card_convention_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_card_params_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_card_voltage_class_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_context_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_control_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_direction_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.SMARTCARD_EMV_ATR_DURATION_ETU (*C macro*), 508
drivers_Kinetis_K_MK22F51212.SMARTCARD_EMV_RX_NACK_THRESHOLD (*C macro*), 517
drivers_Kinetis_K_MK22F51212.SMARTCARD_EMV_RX_TO_TX_GUARD_TIME_T0 (*C macro*), 517
drivers_Kinetis_K_MK22F51212.SMARTCARD_EMV_TX_NACK_THRESHOLD (*C macro*), 517
drivers_Kinetis_K_MK22F51212.SMARTCARD_INIT_DELAY_CLOCK_CYCLES (*C macro*), 508
drivers_Kinetis_K_MK22F51212.SMARTCARD_INIT_DELAY_CLOCK_CYCLES_ADJUSTMENT (*C macro*), 514
drivers_Kinetis_K_MK22F51212.smartcard_interface_callback_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_interface_config_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_interface_control_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_parity_type_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.SMARTCARD_PHY_Activate (*C function*), 514
drivers_Kinetis_K_MK22F51212.SMARTCARD_PHY_Control (*C function*), 514
drivers_Kinetis_K_MK22F51212.SMARTCARD_PHY_Deactivate (*C function*), 514
drivers_Kinetis_K_MK22F51212.SMARTCARD_PHY_Deinit (*C function*), 514
drivers_Kinetis_K_MK22F51212.SMARTCARD_PHY_GetDefaultConfig (*C function*), 513
drivers_Kinetis_K_MK22F51212.SMARTCARD_PHY_Init (*C function*), 513
drivers_Kinetis_K_MK22F51212.smartcard_reset_type_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_time_delay_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_timers_state_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_transfer_callback_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_transfer_state_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.smartcard_transport_type_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.SMARTCARD_TS_DIRECT_CONVENTION (*C macro*), 508
drivers_Kinetis_K_MK22F51212.SMARTCARD_TS_INVERSE_CONVENTION (*C macro*), 508
drivers_Kinetis_K_MK22F51212.SMARTCARD_UART_AbortTransfer (*C function*), 516
drivers_Kinetis_K_MK22F51212.SMARTCARD_UART_Control (*C function*), 516
drivers_Kinetis_K_MK22F51212.SMARTCARD_UART_Deinit (*C function*), 515
drivers_Kinetis_K_MK22F51212.SMARTCARD_UART_ErrIRQHandler (*C function*), 517
drivers_Kinetis_K_MK22F51212.SMARTCARD_UART_GetDefaultConfig (*C function*), 515
drivers_Kinetis_K_MK22F51212.SMARTCARD_UART_GetTransferRemainingBytes (*C function*), 515
drivers_Kinetis_K_MK22F51212.SMARTCARD_UART_Init (*C function*), 515
drivers_Kinetis_K_MK22F51212.SMARTCARD_UART_IRQHandler (*C function*), 516
drivers_Kinetis_K_MK22F51212.smartcard_uart_TimerStart (*C function*), 517

drivers_Kinetis_K_MK22F51212.SMARTCARD_UART_TransferNonBlocking (*C function*), 516
drivers_Kinetis_K_MK22F51212.SMARTCARD_UART_TSExpiryCallback (*C function*), 517
drivers_Kinetis_K_MK22F51212.smartcard_xfer_t (*C type*), 507
drivers_Kinetis_K_MK22F51212.SMC_GetParam (*C function*), 518
drivers_Kinetis_K_MK22F51212.SMC_GetPowerModeState (*C function*), 518
drivers_Kinetis_K_MK22F51212.SMC_GetVersionId (*C function*), 518
drivers_Kinetis_K_MK22F51212.smc_param_t (*C type*), 523
drivers_Kinetis_K_MK22F51212.smc_partial_stop_option_t (*C type*), 523
drivers_Kinetis_K_MK22F51212.SMC_PostExitStopModes (*C function*), 519
drivers_Kinetis_K_MK22F51212.SMC_PostExitWaitModes (*C function*), 519
drivers_Kinetis_K_MK22F51212.smc_power_mode_lls_config_t (*C type*), 523
drivers_Kinetis_K_MK22F51212.smc_power_mode_protection_t (*C type*), 523
drivers_Kinetis_K_MK22F51212.smc_power_mode_vlls_config_t (*C type*), 523
drivers_Kinetis_K_MK22F51212.smc_power_state_t (*C type*), 523
drivers_Kinetis_K_MK22F51212.SMC_PreEnterStopModes (*C function*), 519
drivers_Kinetis_K_MK22F51212.SMC_PreEnterWaitModes (*C function*), 519
drivers_Kinetis_K_MK22F51212.smc_run_mode_t (*C type*), 523
drivers_Kinetis_K_MK22F51212.SMC_SetPowerModeHsrun (*C function*), 519
drivers_Kinetis_K_MK22F51212.SMC_SetPowerModeLls (*C function*), 520
drivers_Kinetis_K_MK22F51212.SMC_SetPowerModeProtection (*C function*), 518
drivers_Kinetis_K_MK22F51212.SMC_SetPowerModeRun (*C function*), 519
drivers_Kinetis_K_MK22F51212.SMC_SetPowerModeStop (*C function*), 519
drivers_Kinetis_K_MK22F51212.SMC_SetPowerModeVlls (*C function*), 520
drivers_Kinetis_K_MK22F51212.SMC_SetPowerModeVlpr (*C function*), 520
drivers_Kinetis_K_MK22F51212.SMC_SetPowerModeVlps (*C function*), 520
drivers_Kinetis_K_MK22F51212.SMC_SetPowerModeVlpw (*C function*), 520
drivers_Kinetis_K_MK22F51212.SMC_SetPowerModeWait (*C function*), 519
drivers_Kinetis_K_MK22F51212.smc_stop_mode_t (*C type*), 523
drivers_Kinetis_K_MK22F51212.smc_stop_submode_t (*C type*), 523
drivers_Kinetis_K_MK22F51212.smc_version_id_t (*C type*), 523
drivers_Kinetis_K_MK22F51212.status_t (*C type*), 364
drivers_Kinetis_K_MK22F51212.SUPPRESS_FALL_THROUGH_WARNING (*C macro*), 367
drivers_Kinetis_K_MK22F51212.SYS_CLK (*C macro*), 164
drivers_Kinetis_K_MK22F51212.TRM (*C macro*), 548
drivers_Kinetis_K_MK22F51212.TWO_BIT_DURATION_MAX_2400 (*C macro*), 376
drivers_Kinetis_K_MK22F51212.TWO_BIT_DURATION_MAX_4800 (*C macro*), 376
drivers_Kinetis_K_MK22F51212.TWO_BIT_DURATION_MAX_9600 (*C macro*), 376
drivers_Kinetis_K_MK22F51212.TWO_BIT_DURATION_MAX_14400 (*C macro*), 376
drivers_Kinetis_K_MK22F51212.TWO_BIT_DURATION_MAX_19200 (*C macro*), 376
drivers_Kinetis_K_MK22F51212.TWO_BIT_DURATION_MIN_2400 (*C macro*), 376
drivers_Kinetis_K_MK22F51212.TWO_BIT_DURATION_MIN_4800 (*C macro*), 376
drivers_Kinetis_K_MK22F51212.TWO_BIT_DURATION_MIN_9600 (*C macro*), 376
drivers_Kinetis_K_MK22F51212.TWO_BIT_DURATION_MIN_14400 (*C macro*), 376
drivers_Kinetis_K_MK22F51212.TWO_BIT_DURATION_MIN_19200 (*C macro*), 376
drivers_Kinetis_K_MK22F51212.UART0_CLK_SRC (*C macro*), 164
drivers_Kinetis_K_MK22F51212.UART1_CLK_SRC (*C macro*), 164
drivers_Kinetis_K_MK22F51212.UART2_CLK_SRC (*C macro*), 164
drivers_Kinetis_K_MK22F51212.UART_Clear9thTransmitBit (*C function*), 527
drivers_Kinetis_K_MK22F51212.UART_ClearStatusFlags (*C function*), 527
drivers_Kinetis_K_MK22F51212.UART_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.uart_config_t (*C type*), 539
drivers_Kinetis_K_MK22F51212.UART_Deinit (*C function*), 525
drivers_Kinetis_K_MK22F51212.UART_DisableInterrupts (*C function*), 528
drivers_Kinetis_K_MK22F51212.UART_DriverIRQHandler (*C function*), 535
drivers_Kinetis_K_MK22F51212.uart_edma_handle_t (*C type*), 544
drivers_Kinetis_K_MK22F51212.uart_edma_transfer_callback_t (*C type*), 544
drivers_Kinetis_K_MK22F51212.UART_Enable9bitMode (*C function*), 526
drivers_Kinetis_K_MK22F51212.UART_EnableInterrupts (*C function*), 528

drivers_Kinetis_K_MK22F51212.UART_EnableMatchAddress (*C function*), 527
drivers_Kinetis_K_MK22F51212.UART_EnableRx (*C function*), 529
drivers_Kinetis_K_MK22F51212.UART_EnableRxDMA (*C function*), 529
drivers_Kinetis_K_MK22F51212.UART_EnableRxFIFO (*C function*), 534
drivers_Kinetis_K_MK22F51212.UART_EnableTx (*C function*), 529
drivers_Kinetis_K_MK22F51212.UART_EnableTxDMA (*C function*), 529
drivers_Kinetis_K_MK22F51212.UART_EnableTxFIFO (*C function*), 534
drivers_Kinetis_K_MK22F51212.UART_GetDataRegisterAddress (*C function*), 529
drivers_Kinetis_K_MK22F51212.UART_GetDefaultConfig (*C function*), 525
drivers_Kinetis_K_MK22F51212.UART_GetEnabledInterrupts (*C function*), 528
drivers_Kinetis_K_MK22F51212.UART_GetInstance (*C function*), 539
drivers_Kinetis_K_MK22F51212.UART_GetRxFifoCount (*C function*), 530
drivers_Kinetis_K_MK22F51212.UART_GetStatusFlags (*C function*), 527
drivers_Kinetis_K_MK22F51212.UART_GetTxFifoCount (*C function*), 530
drivers_Kinetis_K_MK22F51212.uart_handle_t (*C type*), 539
drivers_Kinetis_K_MK22F51212.uart_idle_type_select_t (*C type*), 539
drivers_Kinetis_K_MK22F51212.UART_Init (*C function*), 525
drivers_Kinetis_K_MK22F51212.uart_isr_t (*C type*), 539
drivers_Kinetis_K_MK22F51212.uart_parity_mode_t (*C type*), 539
drivers_Kinetis_K_MK22F51212.UART_ReadBlocking (*C function*), 530
drivers_Kinetis_K_MK22F51212.UART_ReadByte (*C function*), 530
drivers_Kinetis_K_MK22F51212.UART_ReceiveEDMA (*C function*), 543
drivers_Kinetis_K_MK22F51212.UART_RETRY_TIMES (*C macro*), 405, 540
drivers_Kinetis_K_MK22F51212.UART_SendAddress (*C function*), 530
drivers_Kinetis_K_MK22F51212.UART_SendEDMA (*C function*), 542
drivers_Kinetis_K_MK22F51212.UART_Set9thTransmitBit (*C function*), 527
drivers_Kinetis_K_MK22F51212.UART_SetBaudRate (*C function*), 526
drivers_Kinetis_K_MK22F51212.UART_SetMatchAddress (*C function*), 526
drivers_Kinetis_K_MK22F51212.UART_SetRxFifoWatermark (*C function*), 534
drivers_Kinetis_K_MK22F51212.UART_SetTxFifoWatermark (*C function*), 534
drivers_Kinetis_K_MK22F51212.uart_stop_bit_count_t (*C type*), 539
drivers_Kinetis_K_MK22F51212.uart_transfer_callback_t (*C type*), 539
drivers_Kinetis_K_MK22F51212.uart_transfer_t (*C type*), 539
drivers_Kinetis_K_MK22F51212.UART_TransferAbortReceive (*C function*), 533
drivers_Kinetis_K_MK22F51212.UART_TransferAbortReceiveEDMA (*C function*), 543
drivers_Kinetis_K_MK22F51212.UART_TransferAbortSend (*C function*), 532
drivers_Kinetis_K_MK22F51212.UART_TransferAbortSendEDMA (*C function*), 543
drivers_Kinetis_K_MK22F51212.UART_TransferCreateHandle (*C function*), 531
drivers_Kinetis_K_MK22F51212.UART_TransferCreateHandleEDMA (*C function*), 542
drivers_Kinetis_K_MK22F51212.UART_TransferEdmaHandleIRQ (*C function*), 544
drivers_Kinetis_K_MK22F51212.UART_TransferGetReceiveCount (*C function*), 533
drivers_Kinetis_K_MK22F51212.UART_TransferGetReceiveCountEDMA (*C function*), 543
drivers_Kinetis_K_MK22F51212.UART_TransferGetRxRingBufferLength (*C function*), 532
drivers_Kinetis_K_MK22F51212.UART_TransferGetSendCount (*C function*), 532
drivers_Kinetis_K_MK22F51212.UART_TransferGetSendCountEDMA (*C function*), 543
drivers_Kinetis_K_MK22F51212.UART_TransferHandleErrorIRQ (*C function*), 534
drivers_Kinetis_K_MK22F51212.UART_TransferHandleIRQ (*C function*), 534
drivers_Kinetis_K_MK22F51212.UART_TransferReceiveNonBlocking (*C function*), 533
drivers_Kinetis_K_MK22F51212.UART_TransferSendNonBlocking (*C function*), 532
drivers_Kinetis_K_MK22F51212.UART_TransferStartRingBuffer (*C function*), 531
drivers_Kinetis_K_MK22F51212.UART_TransferStopRingBuffer (*C function*), 531
drivers_Kinetis_K_MK22F51212.UART_WriteBlocking (*C function*), 530
drivers_Kinetis_K_MK22F51212.UART_WriteByte (*C function*), 529
drivers_Kinetis_K_MK22F51212.UINT16_MAX (*C macro*), 352
drivers_Kinetis_K_MK22F51212.UINT32_MAX (*C macro*), 352
drivers_Kinetis_K_MK22F51212.UINT64_H (*C macro*), 352
drivers_Kinetis_K_MK22F51212.UINT64_L (*C macro*), 352
drivers_Kinetis_K_MK22F51212.UINTPTR_SIZE (*C macro*), 352

drivers_Kinetis_K_MK22F51212.USEC_TO_COUNT (*C macro*), 354
drivers_Kinetis_K_MK22F51212.vdiv (*C var*), 175
drivers_Kinetis_K_MK22F51212.vref_buffer_mode_t (*C type*), 548
drivers_Kinetis_K_MK22F51212.VREF_CLOCKS (*C macro*), 163
drivers_Kinetis_K_MK22F51212.vref_config_t (*C type*), 548
drivers_Kinetis_K_MK22F51212.VREF_Deinit (*C function*), 545
drivers_Kinetis_K_MK22F51212.VREF_GetDefaultConfig (*C function*), 546
drivers_Kinetis_K_MK22F51212.VREF_GetLowReferenceTrimVal (*C function*), 547
drivers_Kinetis_K_MK22F51212.VREF_GetTrim2V1Val (*C function*), 546
drivers_Kinetis_K_MK22F51212.VREF_GetTrimVal (*C function*), 546
drivers_Kinetis_K_MK22F51212.VREF_Init (*C function*), 545
drivers_Kinetis_K_MK22F51212.VREF_INTERNAL_VOLTAGE_STABLE_TIMEOUT (*C macro*), 547
drivers_Kinetis_K_MK22F51212.VREF_SC_ICOMPEN (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_SC_ICOMPEN_MASK (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_SC_ICOMPEN_SHIFT (*C macro*), 549
drivers_Kinetis_K_MK22F51212.VREF_SC_MODE_LV (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_SC_MODE_LV_MASK (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_SC_MODE_LV_SHIFT (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_SC_REGEN (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_SC_REGEN_MASK (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_SC_REGEN_SHIFT (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_SC_VREFEN (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_SC_VREFEN_MASK (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_SC_VREFST_MASK (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_SC_VREFST_SHIFT (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_SetLowReferenceTrimVal (*C function*), 547
drivers_Kinetis_K_MK22F51212.VREF_SetTrim2V1Val (*C function*), 546
drivers_Kinetis_K_MK22F51212.VREF_SetTrimVal (*C function*), 546
drivers_Kinetis_K_MK22F51212.VREF_TRM_CHOPEN_MASK (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_TRM_CHOPEN_SHIFT (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_TRM_TRIM (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_TRM_TRIM_MASK (*C macro*), 548
drivers_Kinetis_K_MK22F51212.VREF_TRM_TRIM_SHIFT (*C macro*), 548
drivers_Kinetis_K_MK22F51212.WDOG_ClearResetCount (*C function*), 553
drivers_Kinetis_K_MK22F51212.WDOG_ClearStatusFlags (*C function*), 551
drivers_Kinetis_K_MK22F51212.wdog_clock_prescaler_t (*C type*), 555
drivers_Kinetis_K_MK22F51212.wdog_clock_source_t (*C type*), 555
drivers_Kinetis_K_MK22F51212.wdog_config_t (*C type*), 555
drivers_Kinetis_K_MK22F51212.WDOG_Deinit (*C function*), 550
drivers_Kinetis_K_MK22F51212.WDOG_Disable (*C function*), 550
drivers_Kinetis_K_MK22F51212.WDOG_DisableInterrupts (*C function*), 551
drivers_Kinetis_K_MK22F51212.WDOG_Enable (*C function*), 550
drivers_Kinetis_K_MK22F51212.WDOG_EnableInterrupts (*C function*), 551
drivers_Kinetis_K_MK22F51212.WDOG_FIRST_WORD_OF_REFRESH (*C macro*), 553
drivers_Kinetis_K_MK22F51212.WDOG_FIRST_WORD_OF_UNLOCK (*C macro*), 553
drivers_Kinetis_K_MK22F51212.WDOG_GetDefaultConfig (*C function*), 549
drivers_Kinetis_K_MK22F51212.WDOG_GetResetCount (*C function*), 552
drivers_Kinetis_K_MK22F51212.WDOG_GetStatusFlags (*C function*), 551
drivers_Kinetis_K_MK22F51212.WDOG_Init (*C function*), 549
drivers_Kinetis_K_MK22F51212.WDOG_Refresh (*C function*), 552
drivers_Kinetis_K_MK22F51212.WDOG_SECOND_WORD_OF_REFRESH (*C macro*), 553
drivers_Kinetis_K_MK22F51212.WDOG_SECOND_WORD_OF_UNLOCK (*C macro*), 553
drivers_Kinetis_K_MK22F51212.WDOG_SetTestModeConfig (*C function*), 550
drivers_Kinetis_K_MK22F51212.WDOG_SetTimeoutValue (*C function*), 552
drivers_Kinetis_K_MK22F51212.WDOG_SetWindowValue (*C function*), 552
drivers_Kinetis_K_MK22F51212.wdog_test_config_t (*C type*), 555
drivers_Kinetis_K_MK22F51212.wdog_test_mode_t (*C type*), 555

drivers_Kinetis_K_MK22F51212.wdog_tested_byte_t (*C type*), 555
drivers_Kinetis_K_MK22F51212.WDOG_Unlock (*C function*), 552
drivers_Kinetis_K_MK22F51212.WDOG_WCT_INSTRUCITON_COUNT (*C macro*), 555
drivers_Kinetis_K_MK22F51212.wdog_work_mode_t (*C type*), 555
drivers_Kinetis_K_MK22F51212.workMode (*C var*), 175